Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Dissertation

OPC UA Information Model Design

OPC UA Informationsmodellierung für Cyber-Physical Production Systems

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Doktors der technischen Wissenschaften (Dr.techn)

DI Florian PAUKER

Mat.Nr.: 0649099

unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl
Institut für Fertigungstechnik und Photonische Technologien

begutachtet von

Wolfgang Kastner
Institut für Computer Engineering
Treitlstraße 3, 1040 Wien

Manuel Wimmer
Institut für Wirtschaftsinformatik
Altenberger Straße 69, 4040 Linz

Diese Arbeit wurde von der FFG im Rahmen des Projektes OPC4Factory unterstützt.

Ich nehme zur Kenntnis, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Dissertation

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Eidesstattliche Erklärung:

Ich erkläre Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sin in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht.

Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein.

Wien, April, 2019

_____

Unterschrift

Danksagung

An erster Stelle gilt mein Dank meinem Doktorvater Herrn Prof. Burkhard Kittl für seine wissenschaftliche und methodische Unterstützung während der gesamten Bearbeitungsphase meiner Dissertation.

Herrn Prof. Wolfgang Kastner, Herrn Prof. Manuel Wimmer, Herrn Thomas Frühwirth und Herrn Jürgen Mangler danke ich für die zahlreichen und unermüdlichen fachliche Gespräche, Ratschläge und Anmerkungen, die mich auf dem Weg zur fertigen Arbeit immer wieder neue Aspekte und Ansätze entdecken ließen. Auch die vielen nichtwissenschaftlichen und motivierenden Gespräche haben meine Arbeit unterstützt.

Besonders möchte ich an dieser Stelle auch meinen Eltern, die mir das Studium ermöglicht haben, danken. Des weiteren möchte ich meiner Ehegattin Margit für die unermüdliche Stärkung und Motivierung danken, sowie für das stets offene Ohr für meine Gedanken.

Kurzfassung

Produzierende Unternehmen sind heutzutage mit einem Paradigmenwechsel konfrontiert. Die klassische Massenproduktion wird vermehrt durch individuelle Produktion (Mass Customization) abgelöst. Dieser Trend in Kombination mit der immer größer werdenden Produktvielfalt und kürzeren Produktlebenszyklen führt zu sinkenden Losgrößen. Dies stellt herkömmliche Produktionsanlagen und im speziellen die Automation besondere Anforderungen um flexibel auf die Anforderungen des Marktes reagieren zu können.

Zusätzlich gibt es durch den vermehrten Einsatz von Informations- und Kommunikationstechnologie (IKT) eine Vielzahl von neuen Möglichkeiten, um die Produktivität und Flexibilität zu steigern. Das Schlagwort in diesem Zusammenhang sind sogenannte Cyber-Physical Systems (CPS). CPS verbinden physische Objekte mit virtuellen Objekten. Ein spezielles Merkmal ist, dass diese Komponenten in vernetzen Systemen, welche global über das Internet verbunden sein können, miteinander kommunizieren. Umgelegt auf die Produktionstechnik führt der Einsatz von CPS zu Cyber-Physischen Produktions Systemen (CPPS), welche die anfangs genannten Herausforderungen meistern können.

Um diese Kommunikation zwischen CPS realisieren zu können, benötigt diese gewisse Eigenschaften. Die Kommunikation muss einheitlich und mit Bedeutung (Semantik) erfolgen.
Im Produktionsumfeld gibt es eine Vielzahl von Kommunikationslösungen. Die bekanntesten sind OPC UA, MTConnect und MQTT. Aktuell gibt es nur einen Kommunikationsstandard, der die Anforderungen von CPS erfüllen kann. OPC Unified Architecture (OPC UA) ist die Weiterentwicklung des klassischen OPC Standards und wird als Lösung für diese Kommunikation gesehen. Dies zeigt, dass OPC UA auf zwei wesentlichen Konzepten aufbaut. Einerseits definiert es einen Kommunikationskanal, über den Daten übertragen werden können, anderseits besitzt es eine erweiterbares Metamodell, mit dem die Datenstruktur (Semantik) beschrieben werden kann.

Obwohl OPC UA schon seit einigen Jahren im Einsatz ist, ist die weite Verbreitung noch nicht gegeben. Dies ist einerseits den fehlenden Modellen, anderseits der Komplexität des Standards geschuldet (aktuell 13 Teile in der Spezifikation). Um die Modellierung von Informationsmodellen zu vereinfachen beschreibt diese Arbeit einen systematischen Ansatz, der Entwickler bei diesem Prozess unterstützen soll.

Der Ansatz basiert auf Model Driven Architecture (MDA), einem standardisierten Framework zur modellgetriebenen Softwareentwicklung. Grundidee ist es, die Beschreibung des Systems von der zu implementierenden Technologie zu trennen. Dazu gibt es drei Abstraktionsstufen, welche dabei helfen sollen. In der Arbeit wird dieses Prinzip für die Entwicklung von OPC UA Informationsmodellen erweitert. Diese Erweiterung erlaubt es den kompletten Entwicklungsprozess vom Engineering, über das Design bis hin zur Implementierung von OPC UA Informationsmodellen zu unterstützen.

Kern des OPC UA Information Model Design sind UML Modelle, welche für die jeweiligen Phasen ausgewählt wurden. Im OPC UA Information Model Design werden Komponenten und Use-Case Diagramme für die Beschreibung der Systemanforderungen verwendet. Für das plattformunabhängige Design werden Klassen- und Zustandsdiagramme verwendet. Somit kann sowohl das statische als auch das dynamische Verhalten des Systems abgebildet werden.

Um auf die plattformspezifische Technologie, in diesem Fall OPC UA, zu kommen bedarf es einer Transformation der Modelle, d.h. es müssen Modellelemente von UML auf Elemente von OPC UA gemappt werden. Dieses Mapping und die anschließende Transformation werden ebenfalls behandelt. Für die Transformation wird Atlas Transformation Language (ATL) verwendet, welche erlaubt Elemente der Metamodelle miteinander in Zusammenhang zu bringen.
Um eine möglichst einfache und vollständige Transformation zwischen den Modellen zu ermöglichen, wird das OPC UA Metamodell und das UML Metamodell erweitert. Da UML eine generische Beschreibungs- und Modellierungssprache ist, müssen die Fähigkeiten an OPC UA durch Restriktionen angepasst werden.
Ergebnis der Transformation ist ein OPC UA Informationsmodell, welches durch Instanzierung in einen OPC UA Server integriert werden kann und den Adressraum des Servers bildet.

Als Evaluierung werde in dier Arbeit die mit dem OPC UA Information Model Design erstellten OPC UA und UML Modelle gegenübergestellt und der Mehrwert durch Abstraktion aufgezeigt. Die visuelle Größe und strukturelle Komplexität von Unified Modelling Language (UML) Modellen ist deutlich geringer. Auch bei einigen Design Qualitätsmerkmalen ist UML vorteilhatft.

Abstract

Today, manufacturing companies are confronted with a paradigm shift. Classic mass production is increasingly being replaced by individual production (mass customization). This trend in combination with the ever-increasing variety of products and shorter product life cycles leads to decreasing batch sizes. This places special demands on conventional production systems and especially on automation in order to be able to react flexibly to the requirements of the market.

In addition, the increased use of information and communication technologies (ICT) offers a variety of new possibilities to increase productivity and flexibility. The buzzword in this context are so-called Cyber-physical Systems (CPS). CPS connect physical objects with virtual objects. A special feature is that these components communicate with each other in networked systems that can be globally connected via the Internet. Applied to production engineering, the use of CPS leads to Cyber-Physical Production Systems (CPPS), which can master the challenges mentioned above.

In order to realize this communication between CPS, it needs certain properties. Communication must be uniform and with meaning (semantics).
In the production environment there are a multitude of communication solutions. The best known are OPC Unified Architecture (OPC UA), MTConnect and MQTT. Currently, there is only one communication standard that can meet the requirements of CPS. OPC UA is the further development of the classic OPC standard and is seen as a solution for this communication. This shows that OPC UA is based on two essential concepts. On the one hand, it defines a communication channel via which data can be transferred, and on the other hand, it has an extensible metamodel which allow to describe the data structure (semantics).

Although OPC UA has been in use for several years, it is not yet widely used. This is due to the lack of models and the complexity of the standard (currently 13 parts in the specification). To simplify the modelling of information models, this thesis describes a systematic approach to support developers in this process.

The approach is based on Model Driven Architecture (MDA), a standardized framework for model-driven software development. The basic idea is to separate the description of the system from the technology to be implemented. There are three levels of abstraction

that are supposed to help. In this work, this principle is applied for the development of OPC UA Information models extended. This extension allows to support the complete development process from engineering to design to implementation of OPC UA information models.

The OPC UA Information Model Design is based on Unified Modelling Language (UML) models selected for the respective phases. The OPC UA Information Model Design uses components and use case diagrams to describe system requirements. Class and state diagrams are used for the platform-independent design. Thus, both the static and dynamic behavior of the system can be mapped.

To access the platform-specific technology, in this case OPC UA, the models need to be transformed, i.e., model elements must be mapped from UML to elements of OPC UA. This mapping and the subsequent transformation are also discussed. For the transformation ATLAS transformation language (ATL) is used, which allows to link elements of metamodels.
To enable a simple and complete transformation between the models, the OPC UA metamodel and the UML metamodel are extended. Since UML is a generic description and modeling language, the capabilities of OPC UA must be aligned through restrictions. Result of the transformation is an OPC UA information model, which can be integrated into an OPC UA server by instantiation and forms the address space of the server.

The work also compares the OPC UA and UML models created with the OPC UA Information Model Design and shows the added value through abstraction.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

KAPITEL 1

Einleitung

1.1 Ausgangssituation

1.1.1 Trends in der automatisierten Fertigung

Die produzierende Industrie in Europa steht vor einem Wandel. Der durchschnittliche Anteil der Produktionsumsätze gemessen am Bruttoinlandsprodukt beträgt nur etwa 15%. Damit die industrielle Wertschöpfung und Wettbewerbsfähigkeit in Europa gegeben ist, soll der Industrieanteil am Brutto Inlandsprodukt (BIP) auf 20% gesteigert werden [126] (vgl. Abbildung 1.1).

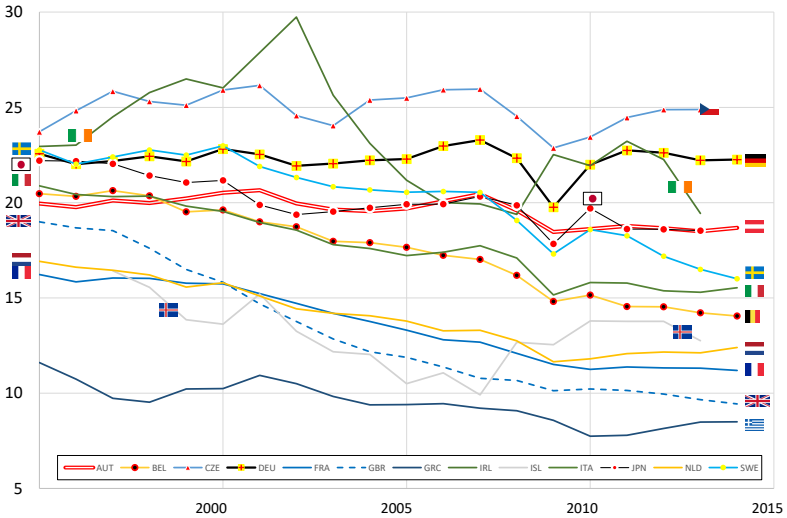

Abbildung 1.1: Added Value Manufacturing (ISIC divisions 15-37) in Europa[1]

[1] https://data.worldbank.org/indicator/NV.IND.TOTL.ZS

Diese Entwicklung begründet sich mit der Tatsache, dass produzierende Unternehmen heutzutage mit einem Paradigmenwechsel konfrontiert sind. Die klassische Massenproduktion wird vermehrt durch individuelle Produktion (*Mass Customization*) abgelöst [42]. Dieser Trend in Kombination mit der immer größer werdenden Produktvielfalt und kürzeren Produktlebenszyklen führt zu sinkenden Losgrößen (Stichwort *Losgröße 1*). Dies hat zur Folge, dass sich Unternehmen verstärkt mit Aspekten wie Flexibilität, Anpassungsfähigkeit, Wandlungsfähigkeit und Rekonfigurierbarkeit in der Fertigung auseinandersetzen müssen. Es ist ein deutlicher Trend Richtung erhöhter Flexibilisierung zu erkennen (siehe Abbildung 1.2) [61].

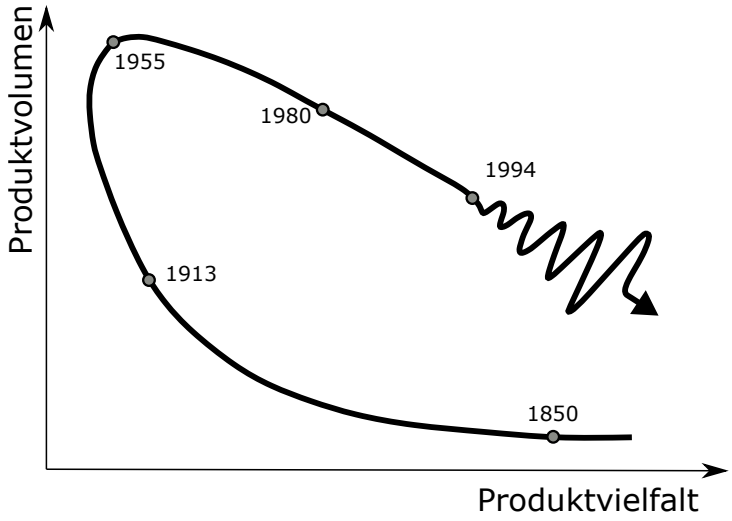

Abbildung 1.2: Fertigungsparadigmen [61].

Hinzu kommen die Weiterentwicklungen im Bereich der Informations- und Kommunikationstechnolgien (IKT). Aktuelle Initiativen, wie Industrie 4.0, Industrial Internet of Things (IIoT) oder ähnliche, streben eine durchgehenden Digitalisierung der Produktion an. Um diese Vision zu realisieren, sollen sogenannte Cyber-Physical Systems (CPS) eingesetzt werden [74]. Die Verwendung dieser CPS in der Fertigung führt zu sogenannten Cyber-Physisches Produktions System (CPPS). CPPS sind gekennzeichnet durch eine Verknüpfung von realen (physischen) Objekten und Prozessen mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze [34].

Die zukünftig erreichbare Flexibilität wird u. a. durch die offene und globale Vernetzung sowie die virtuelle Präsenz der automatisierten Komponenten im Sinne eines CPS deutlich ansteigen. Die Nutzung umfassend verfügbarer Daten, Informationen und Dienste wird

mehr Funktionalität und eine höhere Flexibilität der Anlagenkonfiguration zulassen [13]. Dies soll durch die Wandlungsfähigkeit, Ressourceneffizienz und Ergonomie sowie die Integration von Kunden und Geschäftspartnern in Geschäfts- und Wertschöpfungsprozesse erreicht werden. In intelligenten Fabriken, den Smart Factories, können Produkte wirtschaftlicher hergestellt werden [55].

1.1.2 Vernetzung am Shopfloor

Die aktuelle Herausforderung besteht also darin, aktuelle Fertigungssysteme in CPS zu überführen und den gestellten Anforderungen an eine solche Komponente gerecht zu werden. Dies bedeutet, dass Objekte inklusive ihrer virtuellen Darstellung miteinander vernetzt werden sollen.
Um den Anforderungen an ein CPS zu genügen, müssen die neu geschaffenen Objekte gewisse *self-X* Eigenschaften aufweisen. Diese sind an die *self-CHOP* Eigenschaften, welche von Kephart et al. [59] definiert wurden, angelehnt. Eine dieser Eigenschaften ist die Fähigkeit, in Netzwerken zu kommunizieren und andere Kommunikationspartner zu erkennen (self-organization und context awareness). Eine der wichtigsten Eigenschaften ist die *self-description* Eigenschaft, die eigenen Fähigkeiten zu kennen und anzubieten [128, 143].
Dies bedeutet, dass Sensoren, Maschinen, Anlagen und andere physische Systeme miteinander und mit deren virtuellen Abbildern und anderen virtuellen Objekten und Prozessen wie z.B. Leitsystemen zur Produktionssteuerung und Unternehmensplanung (Manufacturing Execution System (MES) oder Enterprise Resource Planning (ERP)) verbunden werden müssen. Zusätzlich müssen die Systeme über eine semantische Beschreibung verfügen, damit sowohl Menschen als auch andere Systeme die Fähigkeiten der Komponente auslesen können.

Daraus ergibt sich eine Vielfalt an Schnittstellen und Kommunikationsmechanismen, welche für die Realisierung dieser Anforderungen verwendet werden können. Auf dem Markt sind einerseits standardisierte, andererseits proprietäre Lösungen vorzufinden.
Diese Vielfältigkeit und die fehlende Durchgängigkeit der Standardisierung führen letztendlich dazu, dass die Systemintegration aktuell mit hohem Aufwand und erforderlichem Fachwissen verbunden ist (vgl.[57]).

Mit Ethernet existiert bereits seit längerem ein Standard für die physikalische Übertragung und bildet somit den Grundstein für eine standardisierte Kommunikation. Zusätzlich ist mit der aktuellen Spezifikation von Open Platform Communications (OPC) - OPC Unified Architecture (OPC UA) seit einiger Zeit ein Kommunikationsstandard verfügbar, der eine einheitliche und standardisierte Kommunikation zwischen unterschiedlichen Systemen eines Unternehmens ermöglicht. Genau diese Eigenschaften zeichnen OPC UA als derzeit wichtigsten Kandidaten für die Realisierung von CPPS aus.
Obwohl OPC UA seit 2011 als IEC 62541 [47] standardisiert ist, ist der Einsatz und die Verbreitung noch nicht deckend gegeben. Das liegt einerseits an den fehlenden domänenspezifischen Modellen, andererseits an der Skepsis des Nutzens. Beides ist der Komplexität

des Standards geschuldet, der aktuell über 13 Teile mit ca. 1500 Seiten verfügt.

Um die Komplexität zu reduzieren, wird eine umfassende Modularisierung, eine breite Standardisierung sowie eine durchgängige Digitalisierung benötigt [97]. Aktuelle Initiativen (Industrie 4.0, IIoT,...) versuchen genau dieses Ziel zu realisieren.
Die Plattform Industrie 4.0[2] hat dazu das Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) entwickelt (siehe [145]) und mit der Industrie 4.0 Komponente einen Vorschlag für CPS in der Produktion vorgestellt.

1.1.3 Herausforderungen und Reduktion der Komplexität

Im Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) werden die wesentlichen Bestandteile von Industrie 4.0 in einem dreidimensionalen Modell zusammengeführt. Dieses Referenzmodell soll dabei helfen Industrie 4.0 Technologien systematisch einzuordnen und weiterzuentwickeln. Die drei Koordinaten beinhalten die wesentlichen Aspekte von Industrie 4.0 und helfen somit die komplexen Zusammenhänge in kleinere, überschaubare Teilbereiche zu gliedern.
Damit lassen sich die Aufgaben, welche zur Realisierung von Cyber-Physical Systems (CPS) notwendig sind, in kleine überschaubare Aufgaben unterteilen.

Trotz dieses Frameworks stehen viele Firmen vor dem Problem, dass die Entwicklung von CPS und deren Funktionalitäten eine große Hürde darstellen. Dies ist einerseits der Tatsache geschuldet, dass für die Entwicklung Experten aus den Gebieten Produktionstechnik, Maschinenbau, Verfahrenstechnik, Automatisierungstechnik und Informatik benötigt werden [55]. Anderseits sind CPS und CPPS hoch komplexe Systeme und es bedarf eines systematischen Vorgehens, um diese zu reduzieren. Lee et al. haben dazu die Entwicklung in fünf Stufen eingeteilt [66] (siehe Abbildung 1.3).

Für die Bewältigung dieser Komplexität sind neue Methoden und Vorgehensweisen notwendig. Die Verwendung von Modellen hat sich in einigen Domänen (Programmierung bzw. Modellierung von Microcontrollern) als Best-Practice herauskristallisiert, um die Komplexität zu reduzieren [130].

Im Bereich der Modellierung von Systemen gibt es etablierte Vorgehensweisen und Modellierungssprachen. Die wohl bekanntesten Vorgehensmodelle sind das Wasserfallmodell nach Royce [121] sowie das V-Modell [19], eine Weiterentwicklung des Wasserfallmodells. Jedoch bedarf es neuartiger Wege, da im Sinne eines durchgängigen Lebenszyklus kein Unterschied zwischen Design/Implementierung und Operation/Maintenance eines Systems gemacht werden kann. Zusätzlich müssen die Methoden mit einer deutlich gestiegenen Systemkomplexität zurecht kommen [130].

Ein Ansatz für die Lösung dieser Problematik ist die modellgetriebene Softwareen-

[2] http://www.plattform-i40.de/

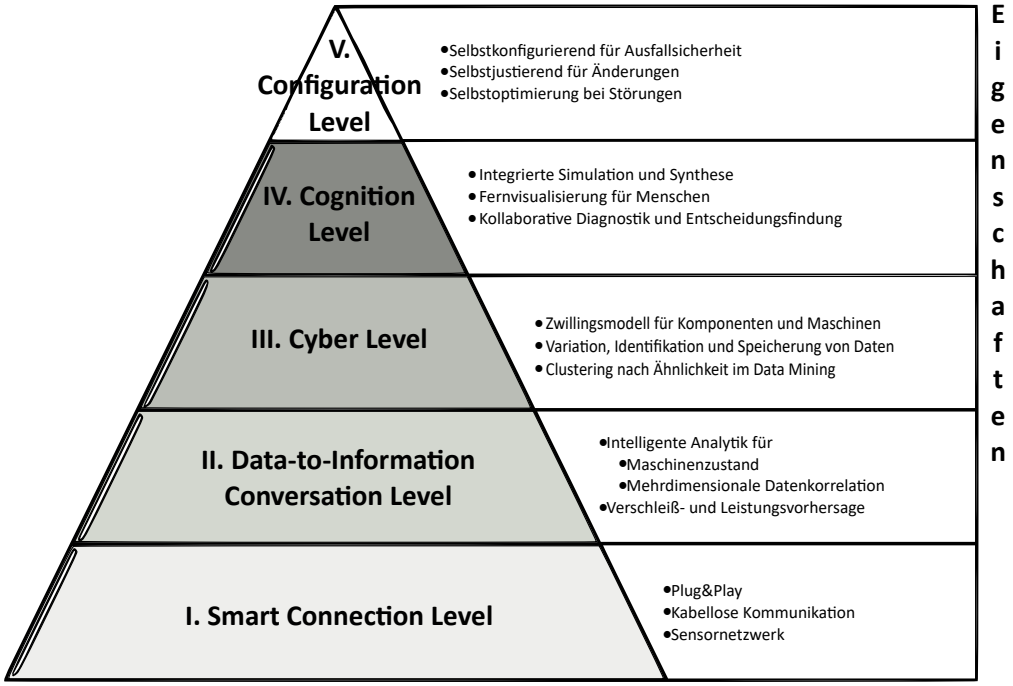

Abbildung 1.3: 5 C Architektur für CPS [66]

twicklung (MDSD). Darunter werden alle Techniken verstanden, welche automatisiert aus formalen Beschreibungen mit Modellen lauffähige Software erzeugen [135].

Ein Problem besteht jedoch, dass bei der Modellierung von Systemen auf unterschiedliche Systemsichten eingegangen werden muss. Für jede Funktion oder Sicht von CPS oder CPPS kann es notwendig sein, eine eigene Modelliersprache zu verwenden. Für die Modellierung von Kommunikationsschnittstellen kann ein OPC UA Modell die geeignete Wahl sein. Für die Beschreibung der einzelnen Komponenten eines System für das Design kann SysML bzw. für das Engineering AutomationML eine geeignete Beschreibungssprache sein. Trotz dieser unterschiedlichen Ansätze entspringen die Modelle vielfach gemeinsamen Systemanforderungen (siehe Abbildung 1.4).

Eine Spezialisierung des MDSD Ansatzes ist der von der Object Management Group (OMG)[3] standardisierte Ansatz Model Driven Architecture (MDA). MDA ist ein Framework für die modellgetriebene Softwareentwicklung mit der klaren Trennung von Systemfunktionalität von der Implementierungs-Technologie [85]. Dafür stellt MDA drei Abstraktionsstufen zur Verfügung: das CIM (Computation Independant Model), das PIM (Platform Independant Model) und das PSM (Platform Specific Model), jedoch wird in MDA nicht beschrieben, welche konkreten UML-Modelle für die jeweilige Domäne

[3]https://www.omg.org/

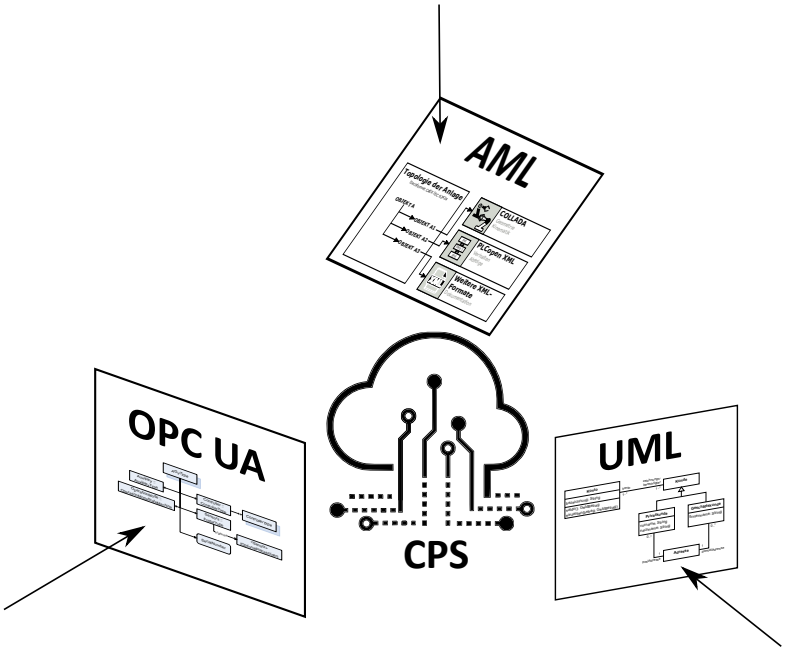

Abbildung 1.4: Systemsichten und Modellierungssprachen

bzw. den jeweiligen Anwendungsfall geeignet sind. MDA setzt dabei auf die Modellierung des Systems mit UML-Diagrammen. Unified Modelling Language (UML) ist die weitverbreiteste Modellierungssprache bzw. Systembeschreibung und kann auf Grund ihrer Fähigkeiten für unterschiedlichste Anwendungsfälle verwendet werden.

1.1.4 Modellgetriebene Erstellung von OPC UA Informationsmodellen

Das Konzept von CPS bzw. der Industrie 4.0 (I4.0) Komponente fordert also eine Beschreibung eines Gegenstandes für andere Teilnehmer in einem Verbund in Form von Eigenschaften inklusive einem dynamischen Verhalten. Zusätzlich ist OPC Unified Architecture (OPC UA) als einziger Kommunikationsstandard genannt, der diese Informationen inklusiver einer Semantik in Form von komplexen Informationsmodellen darstellen kann.

Diese Anforderungen haben zu der Überlegung geführt, die virtuelle Repräsentation einer I4.0 Komponente in Form von OPC UA Informationsmodellen zu realisieren. In Konformität zu der Spezifikation der Verwaltungsschale [1] kann mit Hilfe von OPC UA Informationsmodellen die Beschreibung von Gegenständen erfolgen. Da es auch hier unterschiedliche Sichtweisen, welche unter Umständen mit unterschiedlichen Mod-

ellierungssprachen beschrieben werden, innerhalb einer Verwaltungsschale geben kann, ist es sinnvoll auf eine universelle Sprache für die Modellbeschreibung, wie UML zu setzen.

Im Sinne der Industrie 4.0 Initative, welche das Ziel hat, bestehende Standards zu verwenden, ist die Verwendung des MDA Ansatzes für die Reduktion der Komplexität sinnvoll. Um die Vision von der Vernetzung von Systemen zu realisieren bedarf es der Zusammenarbeit von Experten aus unterschiedlichen Domänen [55]. Darum ist eine einfache universelle Sprache bzw. Modellierung erforderlich. UML bietet alle diese Eigenschaften. Daher ist der MDA Ansatz, welcher UML als Modellierungssprache vorsieht, im Sinne aktueller Bestrebungen.

1.2 Zielsetzung

Ziel der Arbeit ist die Komplexität der Entwicklung von semantischen *Communication Interfaces* für Cyber-Physical Systems (CPS) bzw. Industrie 4.0 Komponenten zu reduzieren. Unter *Communication Interfaces* wird in dieser Arbeit eine auf SOA (Service-Oriented Architecture) basierende Software verstanden, welche die Kommunikation mit anderen Systemen ermöglicht. Hier wird der Fokus auf die Modellierung von Informationen, welche für eine Orchestrierung bzw. Überwachung der einzelnen Komponenten benötigt wird, gelegt.

Konkret sollen OPC Unified Architecture (OPC UA)-Informationsmodelle definiert werden, welche wiederum in OPC UA-Applikationen verwendet werden. Informationsmodelle in OPC UA sind nicht nur einfache Datenmodelle. Es ist ein voll vernetztes graphenbasiertes Netz, welches die Realisierung beliebiger Strukturen ermöglicht.

Mit Hilfe eines modellgetriebenen Softwareentwicklungsansatzes, dem Model Driven Architecture (MDA), soll die Erstellung solcher Modelle vereinfacht werden. Dabei wird das MDA-Framework herangezogen und für diesen Anwendungsfall Information Model Design von OPC UA Informationsmodellen für CPS adaptiert.

Neben der Definition der zu verwendenden Unified Modelling Language (UML) Modelle in den einzelnen Abstraktionsstufen von MDA wird besonders die automatisierte Transformation des Platform Independent Model (PIM) in Platform Specific Model (PSM) und Code berücksichtigt. Dafür werden Transformationsregeln und Mappings vorgestellt, welche es erlauben, UML Diagramme in OPC UA Informationsmodelle zu transformieren.

Anhand eines einfachen Beispiels soll der Ansatz validiert werden und mit Metriken der Vorteil der Abstraktion in plattformunabhängige und -spezifische Modelle gezeigt werden.

1.3 Schwerpunkte und Gliederung der Arbeit

Kapitel 1 liefert eine Motivation sowie die Zielsetzungen für die Erarbeitung des OPC UA Information Model Designs. In Kapitel 2 wird ein Überblick über die aktuellen Bestrebungen und Herausforderungen in der Fertigung gegeben. Besonders wird hier auf CPS und deren Möglichkeiten in der Fertigung eingegangen.

Kapitel 3 widmet sich der Machine-to-Machine (M2M) Communication Lösungen in der Produktion. Dabei wird zu Beginn des Kapitels ein kurzer Überblick über die Entwicklung in der Fertigungstechnik gegeben. In dem folgenden Kapitel wird eine detaillierte Darstellung der aktuell verfügbaren Kommunikationslösungen im Umfeld der Produktion präsentiert. Dabei wird besonderes Augenmerk auf die beiden Technologien OPC UA und MTConnect gelegt, da sich diese für die "semantische" Modellierung eignen und aktuell in Europa und Amerika vermehrte Anwendung finden. Außerdem wird noch auf Message Queue Telemetry Transport (MQTT), einer im Internet of Things (IoT) sehr beliebten Technologie eingegangen.

Kapitel 4 beschäftigt sich mit den Grundlagen der Modellierung. Dabei wird zu Beginn MDA näher vorgestellt. In den folgenden Kapiteln wird auf die UML, als wohl bekannteste Notation, eingegangen und die in der Arbeit verwendeten Modelltypen kurz erklärt. Außerdem wird auch die Notation von OPC UA und deren Verwendung in der Modellierung in diesem Kapitel erläutert.

Der Hauptteil der Arbeit stellt die Methode basierend auf dem MDA Framework vor und beschreibt diese im Detail. Die einzelnen Phasen von den Systemanforderungen bis zur fertigen Software, welche das Information Interface bereitstellt, werden genau erklärt. Neben der Beschreibung der einzelnen Phasen wird besonderes Augenmerk auf die automatisierte Transformation der Modelle gelegt. Dazu werden die Transformationsregeln für die Transformation von UML auf OPC UA werden in Kapitel 6 beschrieben.

Zusätzlich wird in Kapitel 7 eine Evaluierung der Komplexitätsreduktion der Modellierung im Vergleich zu anderen Ansätzen vorgenommen. Anhand der Modellierung eines OPC UA Informationsmodells für ein fahrerloses Transportfahrzeug werden die einzelnen Phasen der Methodik näher erläutert. Dieses konkrete Beispiel soll die Anwendung deutlich machen. In Kapitel 8 werden auch Metriken vorgestellt, anhand derer die Methodik evaluiert und der Vorteil der Modellierung mit UML Modellen und anschließender Transformation in OPC UA Informationsmodelle dargestellt wird.

In Kapitel 9 werden die Ergebnisse noch einmal zusammengefasst und weitere mögliche Entwicklungsschritte aufgezeigt.

KAPITEL 2

Digitalisierung in der Produktion

Die physische und virtuelle Welt verschmelzen zunehmend. Immer mehr physische Objekte verfügen über intelligente Sensor- und Aktuatortechnologien sowie virtuelle Abbildungen und sind meist global miteinander vernetzt. Diese neu entstandenen Cyber-Physical Systems (CPS) liefern alle relevanten Informationen in nahezu Echtzeit, was neue Möglichkeiten in den Anwendungen bewirkt. Die Einführung dieser CPS in die Produktion führt zu Cyber-Physisches Produktions System (CPPS). Abbildung 2.1 zeigt, dass einige Bausteine für die Entwicklung solcher Systeme notwendig sind und dass ein systematisches Vorgehen bei der Entwicklung einzelner Bausteine vorteilhaft ist.

<table>
<tr><td colspan="6">Stufe 2:
Cyber-Physisches Produktionssystem (CPPS)</td></tr>
<tr><td colspan="3">Baustein 1
Maschine zu Maschine Kommunikaiton (M2M)
Über standardisierte Syntax und Dienste</td><td colspan="3">Baustein 3
Mensch Maschine Interaktion (MMI)
Über Schnittstellen wie Virtual Reality oder Augmented Reality</td></tr>
<tr><td colspan="6">Stufe 1:
Cyber-Physisches System (CPS)</td></tr>
<tr><td colspan="3">Baustein 1
Ubiquitous Computing</td><td colspan="2">Baustein 2
Internet der Dinge und Dienste (IoTS)</td><td>Baustein 3
Cloud Computing</td></tr>
<tr><td colspan="3">Mikroelektronik und Sensorik</td><td rowspan="2">Baustein 2.1
Verbindungsstück zwischen Objekten und dem Internet über das Internet Protocol Version 6 (IPV6)</td><td rowspan="2">Baustein 2.2
Bereitstellung von Dienstleistungen über das Internet</td><td>Baustein 3.1
Big Data und Analytics-Dienste</td><td>Baustein 3.2
Bereitstellung von IT-Ressourcen</td></tr>
<tr><td>Baustein 1.1
Intelligente Produktionsmittel</td><td>Baustein 1.2
Intelligente Produkte</td><td>Baustein 1.3
Intelligente Maschinen</td><td colspan="2"></td></tr>
</table>

Abbildung 2.1: Vom CPS zum CPPS [120]

Die nachfolgenden Kapitel beschreiben nun die essentiellen Bausteine von Cyber-Physisches Produktions System.

2.1 Internet of Things (IoT)

Die Entwicklung des Internets ist mittlerweile weit fortgeschritten. Diese lässt sich in vier Phasen unterteilen. Das Web 0 steht für die Vernetzung von Dokumenten. Hierbei handelt es sich um logische „one to one“-Beziehungen, welche einer Datenquelle und eines Datenempfängers entsprechen. Es eröffnete schlichtweg neue Geschäftsmodelle, welche als Web 1.0 bezeichnet werden. Dabei geht es um die Vernetzung der Unternehmen, also von logischen „one to one“ zu „one to many“-Beziehungen. Später kam das Web 2.0, welches Menschen untereinander vernetzt und uns seither in Echtzeit kommunizieren lässt. Hierbei sprechen wir von „many to many“-Beziehungen. Das Web 3.0 oft auch als Semantic Web bezeichnet entsteht, wenn man in das vorhandene Netzwerk nur noch Dinge (technische Objekte) als autonome Internetteilnehmer hinzufügt. Nun sprechen wir auch vom sogenannten Internet der Dinge, welches eine neue qualitätssteigernde technische Zusammenarbeit bringt [11]. Das Internet der Dinge ermöglicht die Vernetzung von eingebetteten Systemen mit den am Produktionsprozess beteiligten Maschinen, Produkten, Systemen und Menschen. Jeder Gegenstand wird mit einer eigenen IP-Adresse versehen und eingebunden [120].
Abgeleitet vom Internet der Dinge (IoT) entstand in der zweiten Phase der technologischen Entwicklung der Begriff Internet der Dinge und Dienste (IoTS). Dieses soll Dienstleistungen, wie z.B. das Bereitstellen von Büchern, Videos gewährleisten und ist somit die Erweiterung des allgegenwärtigen Internets [120]. Momentan wird das Standard Internet Protocol Version 4 (IPv4) verwendet, welches durch sein 32-Bit Format bis zu vier Milliarden Adressen unterscheiden kann. Nachdem in Zukunft immer mehr Objekte sich dem Internet erschließen, wird dieser Standard nicht mehr ausreichen. Das IPv4 wird dem IPv6 weichen müssen, um neuen Platz für bis zu 340 Sextillionen Adressen zu bekommen. Durch diesen Einsatz kann der langfristige Bedarf an IP-Adressen gedeckt werden [120].

2.2 Der Weg zu Cyber Physical Production Systems (CPPS)

2.2.1 Cyber-Physical System

Für den Begriff Cyber-Physical Systems (CPS) gibt es eine Vielzahl von Definitionen, welche sich nicht scharf voneinander unterscheiden lassen. Der Begriff CPS, welcher in der Abbildung 2.1 in der ersten Stufe ersichtlich ist, ist erstmals 2006 durch Helen Gill von der National Science Foundation in den USA aufgekommen und beschreibt die Vernetzung von intelligenten Objekten und Prozessen auf Basis neuer Internettechnologien [127]. Man versteht hierbei eine Kombination von Software und Hardware zu einer Verbindung aus komplexen, intelligenten, physischen Objekten, welche jeweils über eine eigene Identität verfügen [120]. Wie ein schematischer Aufbau eines CPS aussehen kann, ist in der Abbildung 2.2 ersichtlich.

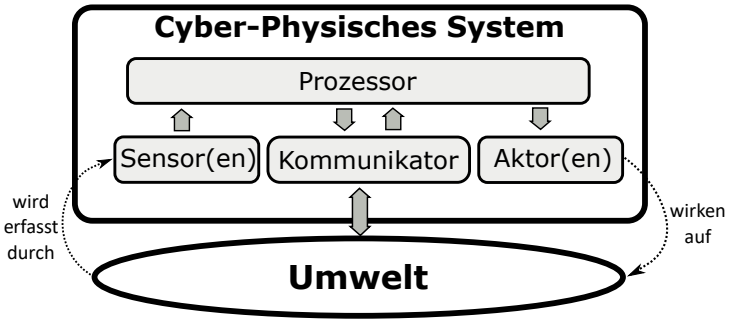

Abbildung 2.2: Schematischer Aufbau eines Cyber-Physical Systems (CPS)[63]

Edward A. Lee führt CPS wie folgt ein [65]:

> „Cyber-Physical Systems (CPS) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa."

Nachfolgend wird für CPS die Definition der Forschungsagenda CPS verwendet [34]. Diese definieren CPS als:

> „Cyber-Physical Systems (CPS) sind gekennzeichnet durch eine Verknüpfung von realen (physischen) Objekten und Prozessen mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze."

Objekte können aus einfachen Modulen bis hin zu ganzen Produktionsanlagen bestehen [127]. Vorerst bleibt eine Frage offen: Gibt es einen Unterschied zu den bestehenden modernen Automatisierungsanlagen? Seit den 70er Jahren existieren eingebettete Systeme bzw. Embedded Systems, wo informationsverarbeitende Komponenten mit physischen Objekten miteinander kommunizieren. Heute ist man überzeugt, dass CPS für die weitreichende Weiterentwicklung der Automatisierungstechnik stehen, denn sie bewirken die Bereitstellung der Informationen aus den vernetzten Subsystemen über das Internet der Dinge [6, 41].

2.2.2 Cyber Physical Production Systems

Kombiniert man nun CPS mit Technologien in der Produktionstechnik, erhält man ein sogenanntes Cyber-Physisches Produktions System (CPPS) [74].

Ein Grundmerkmal ist der hohe Vernetzungsgrad der einzelnen Komponenten und Systeme, der sich bei CPPS ergibt. Nicht nur dieser Vernetzung, sondern auch den

allseits vorhandenen Daten und Diensten wird es zu verdanken sein, dass die Automation neue Dimensionen erreicht. Produktionsanlagen können durch die in Echtzeit übermittelten Daten kostengünstig und ökologisch und dadurch sehr effizient produzieren. Dies macht sich vor allem in der Einsparung der nicht wertschöpfenden Zeiten, aber auch im Ressourceneinsatz bemerkbar [13]. Durch diese Entwicklung können nun Werkzeugmaschinen und Produkte Informationen miteinander austauschen, Fertigungsaufträge anstoßen und andere Komponenten, wie z.B. Automatisierungseinrichtungen, eigenständig steuern. Der Informationsaustausch erfolgt dabei über alle Ebenen der Automatisierungspyramide hinweg, vom Sensor, über einzelne Prozesse und Maschinen bis hin zur Produktions- und -steuerungsebene [13].

Um CPPS zu realisieren, müssen einige Forschungsfragen geklärt werden. Eine der wichtigsten ist die Definition von standardisierten Schnittstellen (siehe Baustein 1 in CPPS in Abbildung 2.1) für eine vertikale und horizontale Vernetzung in der Wertschöpfungskette. Um diese einfach definieren zu können, müssen neue Methoden entwickelt werden [74]. Einen Ansatz hat die Plattform Industrie 4.0 mit dem Referenzarchitekturmodell und der darin vorgestellten Industrie 4.0-Komponente präsentiert.

Die Plattform Industrie 4.0 hat mit dem Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) ein Konzept definiert, wie unterschiedliche Sichtweisen zur Entwicklung von zukünftigen Produktionssystemen und deren Subsystemen zusammengeführt werden können. Bei der Entwicklung von solchen Systemen müssen Experten aus der Prozesstechnik, Fabrikautomation und Informatik miteinander arbeiten sowie unterschiedlichste Standards berücksichtigt werden.

2.2.3 Referenzarchitekturmodell Industrie 4.0

RAMI 4.0 ist ein Schichtenmodell anhand dessen Industrie-4.0-Technologien systematisch eingeordnet und weiterentwickelt werden können. Außerdem ist es behilflich bei der Definition neuer Geschäftsmodelle. RAMI 4.0 basiert auf dem Smart Grid Architecture Model (SGAM) [23] und wurde um Aspekte für Industrie 4.0 erweitert. RAMI 4.0 ist ein dreidimensionales Modell, dessen Dimensionen die vertikale Vernetzung in einer Fabrik, die unterschiedlichen Lebenszyklen des Produktes und hierarchische Schichten (abgeleitet vom SGAM) sind.

Die rechte horizontale Achse zeigt die aus der IEC 62264 [46] - Standard for Enterprise IT and Control Systems bekannten Hierarchielevel. Sie zeigt die einzelnen Entitäten in einer Fabrik gruppiert nach ihrer Funktionalität. Zusätzlich wurde die Unterteilung durch zwei weitere Ebenen erweitert, welche in Industrie 4.0-Anwendungen vorkommen, das *Produkt* und die *Connected World*. Somit können alle Elemente eines Industrie 4.0-Umfelds vom Produkt bis hin zum Internet of Things and Services (IoTS) abgebildet werden. Die horizontale Achse im Vordergrund zeigt die unterschiedlichen Lebenszyklusphasen von Anlagen und Produkten. Die unterschiedlichen Phasen basieren auf dem IEC 62890 - Lifecyclemanagement von Systemen und Produkten der Mess-, Steuer- und

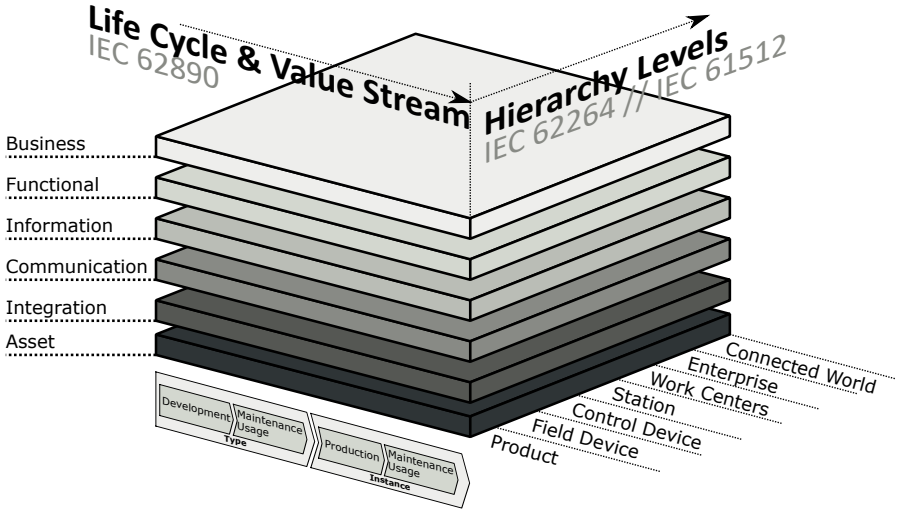

Abbildung 2.3: Referenzarchitekturmodell Industrie 4.0 [144]

Regelungstechnik der Industrie [48]. Hier wird prinzipiell zwischen Typ und Instanz unterschieden. Die Unterscheidung wird eindeutig durch RAMI 4.0 geklärt. Ein Typ wird zur Instanz, wenn die Entwicklung (Design und Prototyping) abgeschlossen ist und das Produkt für die Fertigung freigegeben ist.
Die vertikale Achse beschreibt den Aufbau von Objekten gegliedert in Ebenen (Layer). Diese Darstellungsform stammt ursprünglich aus der Informations- und Kommunikationstechnologie (IKT) Welt, wo komplexe Systeme gerne in Ebenen dargestellt werden, um den Komplexitätsgrad zu reduzieren.

Diese drei Achsen beinhalten nun alle wesentlichen Aspekte von Industrie 4.0. Dies erlaubt es jegliche Objekte wie z.B. Maschinen basierend auf dieser Architektur zu klassifizieren. Somit können die Industrie 4.0 Konzepte mithilfe von RAMI 4.0 beschrieben und umgesetzt werden.

2.2.4 Industrie 4.0-Komponente

Ein konkreter Ansatz für die Verschmelzung von physischer und virtueller Welt in der Produktion ist die Industrie 4.0-Komponente (I4.0-Komponente) [145].

Eine I4.0-Komponente ist ein Modell, welches die Eigenschaften von CPS in der Produktion genauer beschreibt [152].
Das Konzept der Industrie 4.0-Komponente bietet Funktionen auf unterschiedlichen Ebenen an. Diese Ebenen sind in Abbildung 2.4 dargestellt. Die unterste Ebene beschreibt den Gegenstand. Zukünftig muss jeder Gegenstand eine Entität sein, d.h., individuell bekannt und darüber hinaus Daten wie z.B. Lebenszyklusdaten bereitstellen. Auf der nächsten Ebene wird jede Entität in Typ und Instanz unterteilt. Ein Gegenstand bleibt

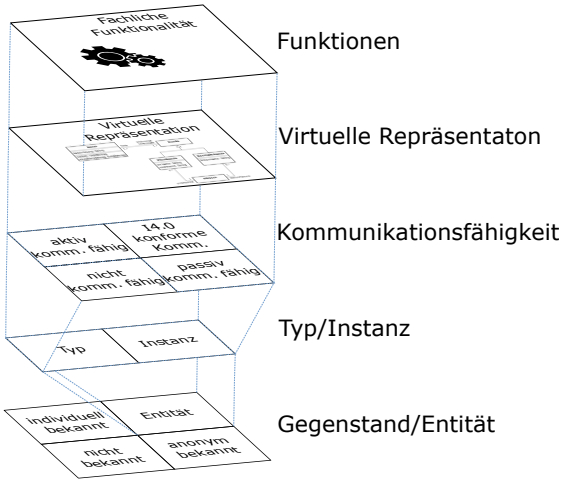

Abbildung 2.4: Beschreibung einer Industrie 4.0-Komponente

solange ein Typ, bis er produziert wird. Erst danach wird der Gegenstand eine Instanz. Die dritte Ebene beschreibt die Kommunikationsfähigkeit der I4.0-Komponente. Hier wird grundsätzlich in vier Kategorien unterschieden. Besitzt ein Gegenstand keine Schnittstellen zur Kommunikation, so ist *nicht kommunikationsfähig*. Solche Gegenstände können nicht in eine I4.0-Komponente umgewandelt werden. Dazu benötigen sie zumindest *passive Kommunikationsfähigkeit*. Ein Gegenstand verfügt über diese, wenn er einen Informationsträger besitzt, der über Schnittstellen ausgelesen werden kann (z.B. RFID Tag). *Aktiv kommunikationsfähig* ist ein Gegenstand, wenn er über eine Steuerung verfügt und über Schnittstellen das Abrufen von Daten erlaubt. *I4.0 konforme Kommunikation* ist die hochwertigste Kommunikationsfähigkeit. Hier stellt der Gegenstand seine Daten und Funktionen nach dem Service Oriented Architecture (SOA) Prinzip inklusive einer I4.0-konformen Semantik zur Verfügung.

Die virtuelle Repräsentation verwaltet Daten zum jeweiligen Gegenstand. Die Daten können, abhängig von der Kommunikationsfähigkeit, von der I4.0-Komponente oder von einem übergeordneten System gespeichert werden. Beide Systeme stellen diese Informationen dann über einen geeigneten Kommunikationskanal der Außenwelt zur Verfügung. RAMI 4.0 sieht vor, dass die virtuelle Repräsentation auf der Informationsebene stattfindet. Somit kommt dem Kommunikationskanal eine hohe Bedeutung zu.
Ein essentieller Bestandteil der virtuellen Repräsentation ist das Manifest. Es ist im Wesentlichen ein Verzeichnis der einzelnen Dateninhalte, Angaben zur I4.0-Komponente an sich (Angaben zur Verbindung mit dem Gegenstand) sowie weitere Daten welche einzelne Lebenszyklusphasen umfassen (z.B. CAD Daten, Handbücher).

Zusätzlich zu Daten besitzt eine I4.0-Komponente auch eine fachliche Funktionalität. Diese ist laut RAMI 4.0 der Funktionsschicht zugeordnet. Beispiele dafür sind: Funktionen

zur ”lokalen Planung” in Verbindung mit dem Gegenstand (Schweißplanung), Funktionen zur Konfiguration, Bedienung bzw. Wartung.

Grundidee ist, dass jeder Gegenstand mit einer Verwaltungsschale erweitert werden kann (siehe Abbildung 2.5) und somit zu einer I4.0-Komponente wird. Diese Software ermöglicht nun jedem Gegenstand die oben genannten Funktionen und Eigenschaften zu vermitteln, egal ob dieser passiv oder aktiv ist, d.h., über eine Steuerung verfügt.

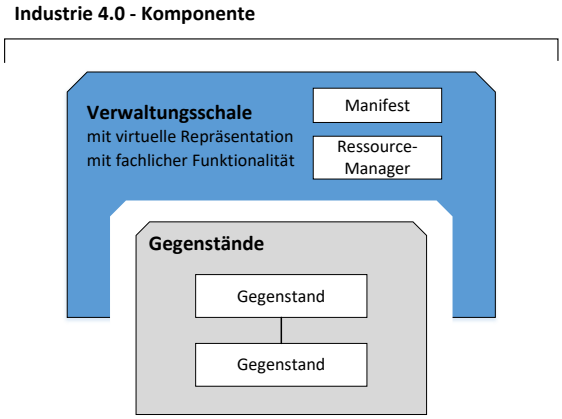

Abbildung 2.5: Industrie 4.0 Komponente

Dabei muss eine I4.0-Komponente laut [145], [152] folgende Merkmale besitzen:

Identifizierbarkeit: Die I4.0-Komponente ist eindeutig im Netz identifizierbar. Die physische Komponente ist mit einer eindeutigen ID beschrieben.

I4.0-konforme Kommunikation: Damit I4.0 Komponenten miteinander interagieren können, können Informationen zwischen ihnen über einen I4.0 konformen Kanal ausgetauscht werden (siehe Abbildung 2.6). Sie unterstützt alle für I4.0 standardisierten Zustände und Dienste. Aktuell gibt es unterschiedliche Ansätze, mit welcher Technologie diese Kommunikation erfolgen kann. Ein Kandidat, der alle Anforderungen erfüllt, ist OPC Unified Architecture (OPC UA).

Schachtelbarkeit Damit die Interoperabilität weiter gesteigert werden kann, können Komponenten verschachtelt werden (siehe Abbildung 2.6), d.h. mehrere Komponenten lassen sich zu einer neuen Komponente kombinieren.

Security und Safety: I4.0-Komponenten müssen grundlegende IT-Security-Anforderungen erfüllen. Zusätzlich können in Anwendungen auch Maßnahmen zur funktionalen Sicherheit (Safety) vorhanden sein.

I4.0-konforme Semantik: Die Komponente muss bei der Kommunikation mit anderen I4.0-Komponenten auf eine feste Semantik zurückgreifen können, welche auch in

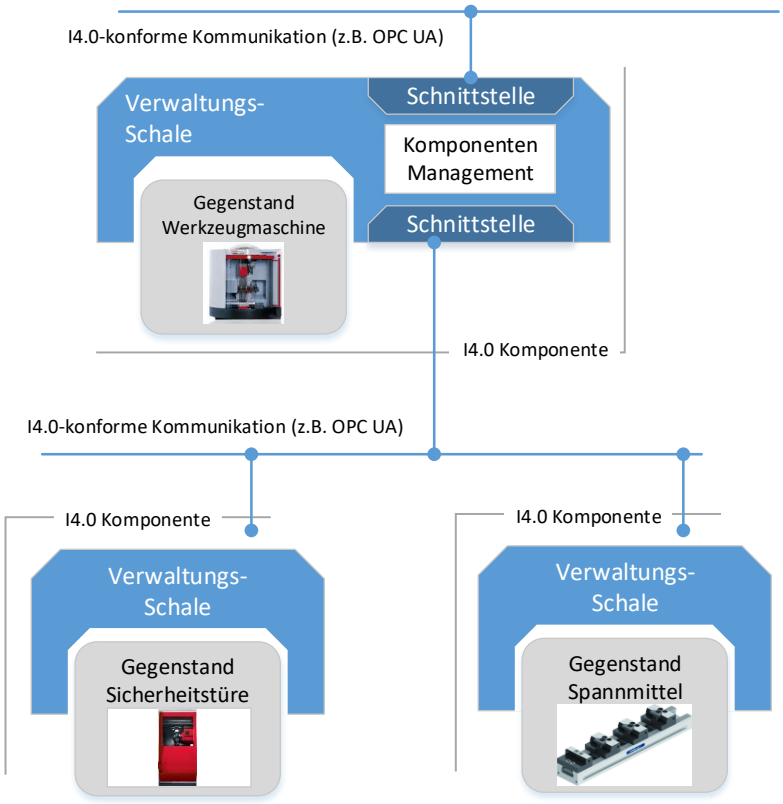

Abbildung 2.6: Interaktion zwischen I4.0-Komponenten

der virtuellen Repräsentation hinterlegt ist und bestimmte Eigenschaften, wie z. B. Funktionalität und Leistungsfähigkeit, beinhaltet.

Quality of Service: Damit ist eine unterschiedliche Behandlung von Daten und Ressourcen hinsichtlich der involvierten Komponenten möglich

Virtuelle Beschreibung: Sie liefert ihre virtuelle Beschreibung inklusive des dynamischen Verhaltens. Diese Beschreibung erfolgt durch die virtuelle Repräsentation und das Manifest.

KAPITEL 3

Kommunikationslösungen in der Fertigung

„Der automatisierte Informationsaustausch zwischen technischen Systemen, wie Maschinen und Geräten, wird als Maschine-zu-Maschine-Kommunikation bezeichnet“ [120].

Der Datenaustausch zwischen zwei oder mehreren Systemen kann sehr gut mit dem von der ISO als IEC 7498-1 [43] standardisierten OSI Referenzmodell beschrieben werden. Dieses regelt auf sieben Schichten (Layern) die Kommunikation zwischen Netzwerkteilnehmern (siehe Abbildung 3.1). Der Abstraktionsgrad der Funktionalität nimmt von Schicht 1 bis zur Schicht 7 zu, wobei die ersten vier eine datenorientierte und die obersten drei die anwendungsorientierte Sicht beschreiben.

Der wichtigste Bestandteile eines Datenaustausches sind die Protokolle. Diese definieren einerseits eine Menge von Regeln zwischen den Kommunikationspartnern, anderseits die Syntax und die Semantik der Kommunikation [112]. Ein Protokoll erfüllt typischerweise eine Funktion und kann daher, abhängig seiner Komplexität eine oder mehrere Schichten des OSI-Modells abdecken. Es bildet somit die Schnittstelle zwischen den unterschiedlichen Ebenen der Datenübertragung. Daraus ergibt sich, dass für die anwendungsorientierten Datenverarbeitung auch nur die relevanten Schichten berücksichtigt werden müssen [150]. Das Protokoll beschreibt somit einen Kommunikationskanal.

Der zweite essentielle Bestandteil eines Datenaustausches ist, dass beide Teilnehmer über eine einheitliche Sprache verfügen. Die Sprache definiert die Syntax und Semantik wird auch als Datenmodell bezeichnet. Ein Datenmodell definiert die Zusammenhänge zwischen einzelnen Daten. Durch Festlegen eines Schemas können diese dann automatisiert interpretiert und verarbeitet werden.

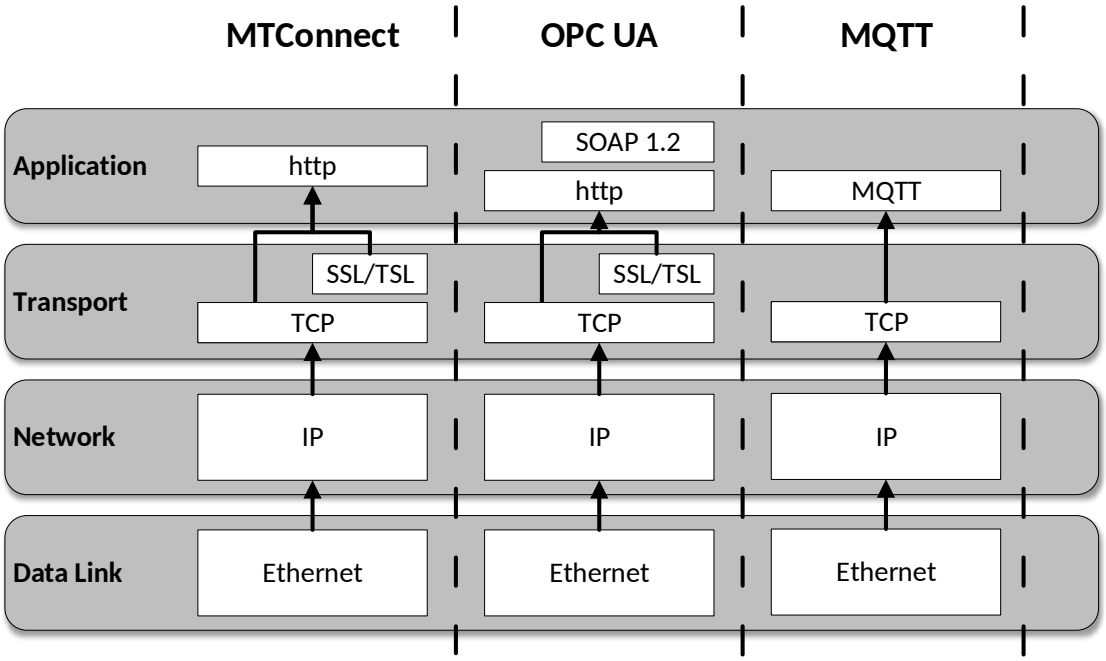

Abbildung 3.1: Kommunikationslösungen im TCP/IP Modell (vgl.[31], [53])

Für jede Datenübertragung definiert jeder Kommunikationskanal ein Datenmodell für den Nachrichtenaufbau. Zusätzlich wird ein weiteres spezifisches Datenmodell für den zu übertragenden Datensatz benötigt. Diese wird im Inhalt einer Nachricht transportiert [150].

Die noch immer sehr weit verbreitetete Datenübertragung in der Fertigung mit Hilfe von analogen und teils digitalen Signalen entspricht nicht dem aktuellen Trend der durchgehenden Digitalisierung.

Einen ersten Schritt in Richtung standardisiertem Datenaustausch bildeten Feldbussysteme. Feldbusse sind bidirektionale, serielle Kommunikationsnetzwerke für die (Echtzeit-) Steuerung von verteilten Automatisierungssystemen. Feldbusse sind für die Vernetzung unterschiedlicher Stellglieder, Aktuatoren und Automatisierungsgeräte gedacht [70]. Trotz der Standardisierungsbemühungen der Feldbussysteme mit der IEC 61158 existiert eine Vielzahl von Feldbus Standards. Die bekannteren sind CAN und ProfiBUS.

Durch die sogenannte *vertikale Integration* bestand jedoch der Bedarf Fertigungssysteme mit anderen Systemen eines Produktionssystems (z.B. Leitrechner, Produktionsplanungs- und Steuerungssysteme) zu verbinden. Die Datendichte wurde größer und die Datenmodelle komplexer. Daher wurde das im Office Bereich verbreitete auf TCP/IP basierte Ethernet auch in der Produktion eingeführt [150].

Für den Zugriff auf Steuerungsdaten bzw. Dateiaustausch haben sich einige Lösungen etabliert. Neben herstellerspezifischen Lösungen wie Sinumerik-RPC und Fanuc FOCAS hat sich OPC etabliert. Die aktuelle Spezifikation von OPC, OPC UA wird als Wegbereiter zur Digitalisierung gesehen, da es eine einheitliche Kommunikation über alle Ebenen im Unternehmen erlaubt [96]. Neben OPC UA gib es noch das aus dem IoT bekannte Message Queue Telemetry Transport (MQTT) und das in den USA weit verbreitete MTConnect Protokoll [68].

Zwischen Softwaresystemen auf Enterprise Level basiert der Datenaustausch meist Dateibasiert, direkt über Abfragen von Datenbanken sowie über REST (Representational State Transfer) Interfaces. Neben den am Shopfloor üblichen 1:1 Client-Server Konzepten kommen hier meist neue Ansätze wie Webarchitecture, Service-oriented Architecture und Multi-Agenten Systeme zum Einsatz [7].

3.1 OPC Unified Architecture - OPC UA

OPC-Unified Architecture (OPC-UA) ist ein hersteller- und plattformunabhängiger Kommunikationsstandard für industrielle Kommunikation. Er stellt die Weiterentwicklung des von der OPC Foundation[1] bereitgestellten Standards OLE for Process Control (OPC) dar und ist als IEC 62541 [47] standardisiert.

Die erste Version wurde 1995 mit dem Ziel veröffentlicht, einen vereinheitlichten Lese- und Schreibzugriff OPC DA [92] auf Prozessdaten von herstellerspezifischen Human Machine Interface (HMI), Supervisory Control and Data Acquisition (SCADA), Netzwerk- und Gerätetechnologien zu ermöglichen. Die Idee dahinter war, dass Hersteller verschiedener Technologien OPC Treibermodule zur Verfügung stellen, die die Verknüpfung mit dem OPC Application Program Interface (API) herstellen. Somit ist gewährleistet, dass Geräte, die auf unterschiedlichen Technologien basieren, untereinander über eine einheitliche OPC-Schnittstelle Daten austauschen können, und nicht jede Technologie Schnittstellen für alle Übrigen bereitstellen muss. Neben dem Zugriff auf aktuelle Daten wurden auch Mechanismen für den Zugriff auf historische Daten OPC HDA sowie für den Zugriff auf Alarms und Events (OPC A&E) integriert.

Schließlich zeigte aber die klassische OPC-Spezifikation gewisse Schwächen, wie die Darstellung komplexer Daten, die Einschränkung auf Windows-Betriebssysteme und fehlende Security-Mechanismen. Um diesen Nachteilen zu begegnen, wurde OPC UA entwickelt. OPC UA vereinigt die Funktionalitäten der Vorgängerversionen und bietet zusätzlich folgende Features:

- Verwendung von Webservices und TCP-basierenden Protokollen zur plattformunabhängigen Kommunikation

[1] https://opcfoundation.org/

- Bereitstellung von leistungsfähigen Security-Mechanismen
- Objektorientiertes Modell zur Abbildung von Systemen jeglicher Struktur und Komplexität
- Abstraktes Metamodell, von dem spezifische Modelle abgeleitet werden, die wiederum das Basismodell erweitern (siehe Abbildung 3.2)
- Prozessdaten können mit zusätzlicher Semantik versehen werden, beispielsweise mit physikalischen Einheiten oder Aussagen über Datengüte

Um diese Anforderungen umsetzen zu können wurde eine neue Architektur konzipiert.

Abbildung 3.2 zeigt den Aufbau sowie die Modellhierarchie von OPC UA.
Basis bilden die Transportmechanismen und die Datenmodellierung [71]. Die Transporttechnologie definiert, wie Informationen ausgetauscht werden, während das OPC UA Metamodell bestimmt, welche Informationen ausgetauscht werden.
Aktuell spezifiziert OPC UA zwei Protokolle. Ein auf TCP basierendes Binärprotokoll für höchste Performance und effiziente Nutzung von Ressourcen, geeignet für embedded Systeme, und Webservices [137]. Somit gibt es genügend Möglichkeiten der plattformunabhängigen Integration von OPC UA Anwendungen in Unternehmen. Die beiden Technologien können durch weitere ergänzt werden. Aktuelle Bemühungen der OPC-Foundation gehen in die Implementierung eines echtzeitfähigen Protokolls basierend auf TSN (Time Sensitive Networking) [109].

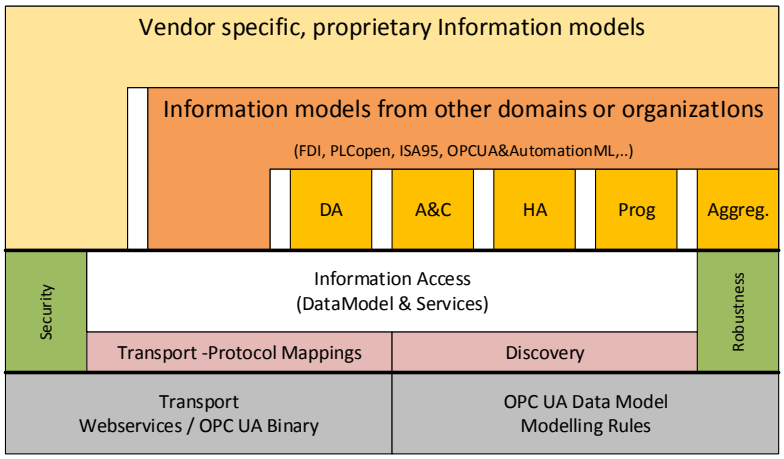

Abbildung 3.2: OPC UA Modellhierarchie [72]

Das Metamodell in OPC UA, dargestellt als OPC UA Data Model in Abbildung 3.2 ist in Teil 3 der Spezifikation beschrieben [100]. Es stellt die Basiselemente, aus denen schließlich Informationsmodelle gebildet werden, zur Verfügung und definiert Regeln, wie

dies zu geschehen hat. Während Classic OPC über ein sehr einfaches Metamodell verfügt, durch das sich nur eine flache "Tag"basierte Hierarchie abbilden lässt, ermöglicht OPC UA die Modellierung von komplexen Informationsmodellen basierend auf der objektorientierten Modellierung. Dies ist dem umfangreichen Metamodell von OPC UA geschuldet.

Darauf aufbauend spezifiziert OPC UA eine abstrakte Sammlung von Basisdiensten (Services) [101] und deren Umsetzung auf konkrete Technologien [103], welche die Basis für die OPC UA Funktionalität darstellen.

Aufbauend auf den Diensten stellt das Basis OPC UA Informationsmodell [102] Basistypen, sowie die Zugangspunkte für den Adressraum des Servers bereit. Das Informationsmodell ist eine Modellinstanz des OPC UA Metamodells. Als weitere Teile von OPC UA sind die Informationsmodelle für Data Access, Alarms and Conditions, Programs und Historical Access zu sehen. Diese stellen eine Integration von Teilen der klassischen OPC Spezifikation in OPC Unified Architecture dar (daher auch der Name) und liefern Mechanismen zur Abbildung von aktuellen Daten, Events, Prozeduren und historischen Prozessdaten.

Darauf aufbauend werden die sogenannten OPC UA Companion Specifications definiert (in Abbildung 3.2 als „Information models from other domains or organizations“ bezeichnet). Darunter sind domänen- bzw. technologiespezifische Modelle zu verstehen, die Vorschriften zur Abbildung von Daten- und Applikationsmodellen aus dem jeweiligen Bereich nach OPC UA beschreiben. Dies soll gewährleisten, dass herstellerübergreifende Interoperabilität durch ein einheitliches, standardisiertes OPC UA Informationsmodell einer bestimmten Zieltechnologie gegeben ist. Damit wird vermieden, dass verschiedene Ausrüstungshersteller ihre individuellen Informationsmodelle verwenden. Dennoch erlauben diese Companion Specifications genügend Freiheiten für proprietäre Erweiterungen, die für herstellerspezifische Gerätefeatures notwendig sein können. Da das Interesse an OPC UA immer größer wird, steigt die Anzahl der Companion Specifications stetig. Beispiele für bereits existierende Companion Specifications sind OPC UA for Devices [93], OPC UA for ISA-95 Common Object Model [94], OPC UA for IEC 61131-3 (PLCopen) [111] sowie für MTConnect [82] und NC Interfaces für Werkzeugmaschinen [146].

Die oberste Stufe in der in Abbildung 3.2 gezeigten Modellhierarchie stellen schließlich die proprietären Erweiterungen dar. Hier erhalten Hersteller die Möglichkeit, über die Companion Specification hinausgehende Funktionalitäten in OPC UA abzubilden, sind aber gleichzeitig dazu angehalten, die Regeln der unteren Schichten in der Modellhierarchie zu befolgen.

OPC UA basiert auf Service Oriented Architecture (SOA). Die Architektur besteht in der Regel aus einem Server, der seine Services im Netzwerk mit Hilfe eines Adressraumes (AddressSpace) anbietet und einem oder mehreren Clients, der auf diesen zugreift. Werden Server und Client in einer Applikation verbunden, so kann ein Server mit anderen

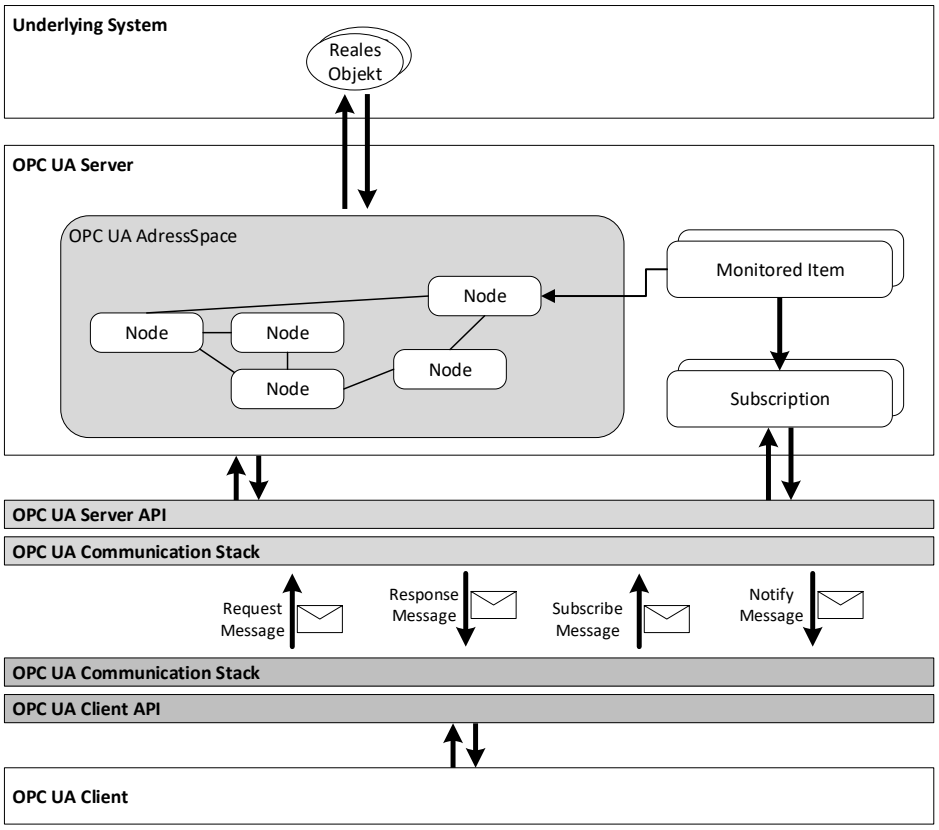

Abbildung 3.3: Überblick OPC UA Konzepte vgl. [104]

Servern interagieren. Die Architektur ist in Abbildung 3.3 dargestellt und in Teil 1 der Spezifikation beschrieben [104]. Die Abbildung zeigt die Interaktion zwischen einem Server und einem Client und stellt die wichtigsten Elemente dar.

Client Der OPC UA Client ist jener Code, welcher für die Implementierung der Client Funktionalität benötigt wird. Die Applikation verwendet ein Client API, welches Service Requests und Responses senden bzw. empfangen kann. Die in OPC UA verfügbaren Services sind in der Spezifikation Teil 4 beschrieben [101]. Das Client API wird dazu verwendet, um die Applikation vom Communication Stack zu trennen. Der Communication Stack wandelt die Anfragen des Client APIs in Nachrichten um und sendet diese über den darunter liegenden Kommunikationskanal. Neben Request und Response Nachrichten verwaltet der Communication Stack auch Subscribe und Notification Messages.

Server Als OPC UA Server wird jener Code bezeichnet, welcher die Funktionen eines Servers erfüllt. Der OPC UA Server verwendet ein OPC UA Server API um

Nachrichten vom Client zu empfangen bzw. diesem welche zu senden. Wie beim Client funktioniert das Server API als Schnittstelle zwischen der Applikation und dem Communication Stack.
Das *Underlying System* beinhaltet *reale Objekte*. Diese sind sowohl reale Objekte als auch Software Objekte, welche über den OPC UA Server verfügbar gemacht werden. Der Adressraum (*AddressSpace*) repräsentiert nun diese realen Objekte, deren Definition und Referenzen zwischen den Objekten. Diese Repräsentation erfolgt mit der Darstellung als *Nodes*, welche mit *References* miteinander verbunden sind. Der Adressraum ist durch OPC UA Services für den Client zugänglich.
Monitored Items sind Entitäten, welche durch den Client angelegt werden. Diese überwachen Knoten im Adressraum und deren reales Gegenstück. Sobald sich die Daten ändern oder bestimmte Ereignisse auftreten wird eine Benachrichtigung (*Notification*) generiert, welche über eine *Subscription* an den Client übertragen wird. Eine *Subscription* ist ein Endpunkt des Servers, welcher Nachrichten an Clients verteilt.
Grundsätzlich wird in OPC UA zwischen Request und Response Services, sowie Publisher Services unterschieden. Request/Response Service werden durch den Client angestoßen und führen eine spezifische Aufgabe bei einem oder mehreren Knoten im Adressraum aus. Der Client erhält vom Server eine Antwort (*Response*). Publisher Services werden dazu verwendet, um Nachrichten an Clients zu senden. Diese Nachrichten sind z.B. Events, Alarme, Änderungen von Daten und Programmrückmeldungen.

3.2 MTConnect

In den USA, vor allem in Nordamerika, hat sich MTConnect etabliert. MTConnect ist ein offenes, erweiterbares und lizenzfreies Kommunikationsprotokoll, entworfen für den Datenaustausch von Shopfloor Equipment mit Software Anwendungen. Der Standard wird vom MTConnect Institute[2] verwaltet, einer Nonprofit-Organisation mit dem Ziel der Verbesserung der Nutzung von Daten in der Fertigungsbranche [75].

MTConnect basiert auf XML und HTTP und benutzt ein REST-Interface für die Kommunikation. Dies hat zur Folge, dass eine große Anzahl an Werkzeugen für die Implementierung vorhanden sind.

In der aktuellen Version (Juni 2015) besteht MTConnect aus vier Teilen. Der erste Teil enthält allgemeine Informationen und bietet eine Übersicht [76]. Der zweite Teil beschreibt das Metamodell für die Modellierung von Devices [77]. Teil drei enthält das Metamodell für die zu übertragenen Daten und damit die Organisation des Datenflusses, d.h. die Kommunikation zwischen den Netzwerkkomponenten [78]. Im Teil 3.1 der Spezifikation werden sogenannte Interfaces definiert. Diese Interfaces erlauben es Requests

[2] www.mtconnect.org

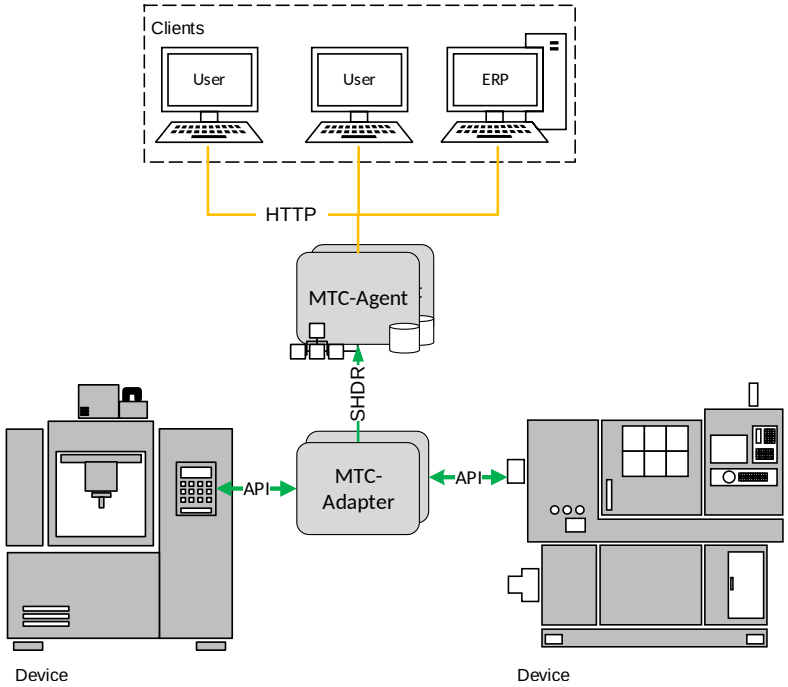

Abbildung 3.4: MTConnect Architektur vgl. [139]

an den MTConnect Agent zu senden [79]. Im vierten Teil wird noch Auskunft über die Verwendung beziehungsweise Modellierung von *Assets* gegeben [80]. Teil 4.1 beschreibt im speziellen die Modellierung von *CuttingTools* [81]. Diese Beschreibung orientiert sich an der ISO 13399 - Werkzeugdatendarstellung und -austausch [50].

Die MTConnect Architektur ist relativ einfach aufgebaut (siehe Abbildung 3.4) [76] Grundsätzlich benötigt man ein Gerät (Device), welches seine Daten in einem Netzwerk bereitstellt, einen MTConnect Agent, welcher als Zwischenspeicher für die Daten des Geräts im Netzwerk dient, und einen Client, der die Daten vom Agent abrufen kann und dem Anwender präsentiert. Im folgenden werden die Funktionen der einzelnen Teilnehmer näher beschrieben.

MTConnect Adapter Der Adapter ist eine optionale Software, welche als gerätespezifische Schnittstelle zwischen Agent und dem Device/Gerät dient. Die Aufgabe des Adapters ist es, die Daten des Gerätes in das Simple Hierarchical Data Representation (SHDR) Protokoll umzuwandeln und diese dann über einen Kommunikationskanal an den Agenten zu übermitteln [30], [75], [138].

MTConnect Agent Die Aufgaben des Agenten sind Daten, die ihm über das Netzwerk angeboten werden (z.B. über einen Adapter) zu speichern, aufzubereiten und in eine standardisierte Form zu bringen. Somit kann er die Daten für Anfragen (Requests) von Clients bereitstellen. Die Speicherung der Daten erfolgt in einem sogenannten

Ringspeicher als Puffer. Dieser kann eine begrenzte Anzahl an Datenelementen zwischenspeichern. Sollte der Puffer voll sein, so werden die ältesten Datensätze gelöscht. Der Agent soll lediglich eine standardisierte Schnittstelle zwischen Device und Anwender bzw. Software sein [75].

MTConnect Client Der Client fordert Daten über eine REST Schnittstelle an und stellt sie einer Anwendung bzw. einem Anwender bereit.

MTConnect Network Das Netzwerk verbindet Devices, Agents, Adapter und Clients miteinander und ermöglicht eine Kommunikation. Die komplette Kommunikation erfolgt über TCP/IP. Die Schnittstelle zwischen Agent und Client nutzt das HTTP Protokoll, während zwischen Adapter und Agent das SHDR zum Einsatz kommt [30].

MTConnect Device Bei dem MT Device handelt sich um ein Gerät, welches Aktionen ausführen kann. Es besteht aus mehren Komponenten, welche während der Ausführung der Aktion Daten liefern. Das Device ist eine eigenständige Entität und muss mindestens einen *component* oder *data item* Datenpunkt bereitstellen [75].

3.3 Message Queue Telemetry Transport - MQTT

MQTT ist ein Client-Server Publish/Subscribe Transportprotokoll und hat sich neben Hypertext Transfer Protocol (HTTP) zu einem der wichtigsten Protokolle im Bereich IoT etabliert. Die wesentlichen Eigenschaften von MQTT sind (i) eine einfache Implementierung, (ii) sicher Übertragung in instabilen Netzwerken, und (iii) die Übertragung von Datentypen unterschiedlichster Art. Diese Eigenschaften erlauben einen weiten Einsatz wie z.b. Machine-to-Machine Kommunikation (M2M) und den Einsatz im Internet of Things (IoT), wo ein effiziente Ressourcennutzung gefordert ist [84, 8].
Seit 2014 ist MQTT über die Organization for the Advancement of Structured Information Standards (OASIS) als IoT-Protokoll standardisiert. Seit 2016 ist MQTT in der Version 3.1.1 als ISO/IEC 20922 genormt [51]. In der Pub/Sub Spezifikation von OPC UA wird MQTT als Transportschicht vorgeschlagen [110].

MQTT folgt dem Publish/Subscribe Pattern (Pub/Sub) und stellt eine Alternative zum klassischen Server-Client Modell dar. Pub/Sub ist ein Nachrichtenentwurfsmuster, wo Sender von Nachrichten (Publisher) Nachrichten nicht gezielt an einen Empfänger(Subscriber) schicken. Sie charakterisieren die Nachrichten in Klassen und veröffentlichen diese unabhängig davon, ob Empfänger vorhanden sind. Genauso bekunden Empfänger Interesse an einer bestimmten Klasse von Nachrichten. Sender und Empfänger müssen nicht direkt miteinander in Kontakt stehen. Ein dazwischengeschalteter Broker verwaltet und vermittelt die Informationen. Abbildung 3.5 stellt die Architektur von MQTT, sowie die wichtigsten Konzepte dar.

Kernstück einer jeden Kommunikation im MQTT ist ein sogenannter *Broker*. Jede

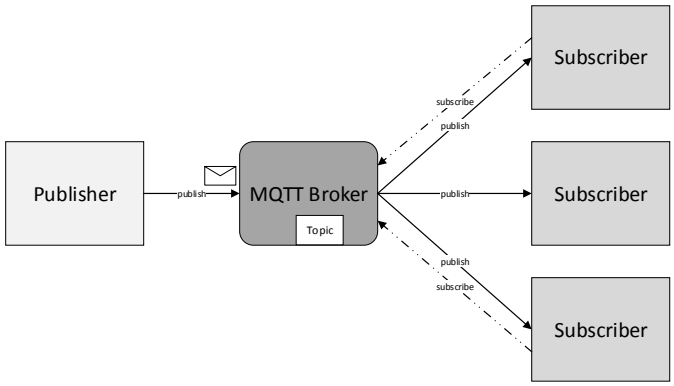

Abbildung 3.5: MQTT Architektur

Nachricht, welche von einem Publisher gesendet wird (*Publish*), besitzt einen Betreff, ein *Topic*. Subscriber können Topics abonnieren. Die Aufgabe des Brokers ist es, die Nachrichten, welche gesendet werden, an die richtigen Empfänger zu verteilen. Die Verteilung der Nachrichten erfolgt somit nach dem *Push* Prinzip und Geräte müssen nicht mittels Polling beim Broker nach neuen Daten fragen. Im nachfolgenden Abschnitt werden die einzelnen Konzepte näher beschrieben.

Client In MQTT werden als Client sowohl Publisher, als auch Subscriber bezeichnet. Grundsätzlich kann in MQTT ein Client beides zur selben Zeit sein. Ein Client kann jedes Gerät, vom Mikrocontroller bis zum Server sein, wo eine MQTT Library läuft und welches über ein Netzwerk mit einem MQTT Broker verbunden ist. MQTT ist ein binäres Protokoll und erfordert normalerweise einen fixen Header von 2 Bytes mit kleinen Nachrichtengröße von bis zu 256 MB [83]. Dies ermöglicht auch einen Einsatz auch auf Embedded Systemen. Die MQTT Libraries sind in einer Vielzahl von Sprachen verfügbar, von Android über Arduino bis hin zu C++ und C#.

Broker Das Gegenstück zum Client ist der sogenannte MQTT Broker, welcher das Herzstück eines Publish/Subscribe Protokolls darstellt. Ein einzelner Broker kann, abhängig von der Implementierung, tausende verbundene Clients besitzen. Die Aufgabe des Brokers ist es, alle Nachrichten zu empfangen, diese zu filtern und die Nachrichten an die daran interessierten, subscribierten Clients zu schicken. Zusätzlich verwaltet der Broker die Sitzung zu allen Clients inklusive deren Subscription und verpassten Nachrichten. Eine weitere Aufgabe des Brokers ist die Authentifizierung und Autorisierung von Clients. Hier setzt MQTT auf bestehenden Mechanismen wie z.B. x509 Client Zertifikate. Zusammengefasst ist der Broker ein zentraler Hub, bei dem jede Nachricht empfangen und verteilt wird. Daher ist es enorm wichtig, dass der Broker skalierbar, einfach zu integrieren und fehlerresistent sein muss.

Topic Unter Topic wir in MQTT jeder Endpunkt verstanden zudem sich ein Client verbinden kann. Jeder Client kann selbst entscheiden, zu welchem Topic er publishen

oder subscriben möchte. Topics sind frei wählbare Zeichenketten und können mittels Slash („/“) hierarchisch aufgebaut werden. Beispiele wären Maschine/FeedOverride und Maschine/Status. Der Eltern-Topic wäre hier Maschine und die untergeordneten Topics wären FeedOverride und Status. Ein Client könnte entweder ein konkretes Kind-Topic oder mittels einer Wildcard („#“) alle Kind-Topics eines Eltern-Topics abonnieren.

3.4 Vergleich M2M Technologien

Zusammenfassend werden in Tabelle 3.1 die Eigenschaften der einzelnen Technologien gegenübergestellt und bewertet.

Eigenschaft	OPC UA		OPC Classic		MTConnect		MQTT	
Interoperabilität	hoch wegen Basismodell	●	nur Windows	◔	Adapter	◑	Pub Sub	◕
Skalierbarkeit	Individuell	●	einzelne Spezifikation	◔	nur Shopfloor	◑	beinahe beliebig	●
Sicherheit	mehrere Konzepte	●	individuell	◑	HTTPS möglich	◑	nicht im Protokoll	○
Transportfähigkeit	mehrere Technologien	◕	COM Technologie	◔	HTTP	◕	Hohe Perfomance	●
Modellierung	komplexe Modelle	●	einzelne Spezifikation	◔	starres Modell	◑	beliebige Strukturen	◕
Erweiterbarkeit	Pub Sub Discovery	●	ausgeprägt	◑	Agent-funktion	◕	Pub Sub	●
Konformität	mehrstufige Tests	●	mehrstufige Tests	◕	nicht vorhanden	◔	offener Standard	◑
Bekanntheitsgrad	steigt in den letzen Jahren	●	in Europa hoch	◕	primär in den USA	◑	IoT	◕
Verwendung in Unternehmen	Clients und einfache Server	◑	nimmt ab wegen OPC UA	◔	in den USA	◑	für Sensoren, einfache Geräte	◑

Tabelle 3.1: Gegenüberstellung der M2M Technologien

Die Kriterien für die Bewertung sind:

Interoperabilität beschreibt die Fähigkeit der Kommunikation zwischen verschiedenen Systemen. Bei diesem Kriterium punkten OPC UA und MQTT wegen der Möglichkeit, auf unterschiedlichen Systemen eingesetzt zu werden.

Skalierbarkeit ist die Anpassungsfähigkeit bezüglich des Umfangs der zu übertragenden Daten und der Anzahl der Komponenten im System. Auch hier sind wieder OPC UA und MQTT von Vorteil, da bei beiden sowohl einfache Strukturen als auch komplexe Datenstrukturen übertragen werden können.

Transportfähigkeit meint die Leistungsfähigkeit des Datentransports, konkret die Übertragungsraten und Datenformate. MQTT ist hier am stärksten, da es hohe Übertragungsraten und beliebige Datenformate garantiert.

Modellierung beschreibt die Fähigkeit, eine große Menge von Daten zu verknüpfen sowie zu strukturieren. Das Meta-Modell von OPC UA sowie dessen Erweiterbarkeit machen OPC UA für die Modellierung von Daten ideal.

Erweiterbarkeit steht für eine schnelle und einfache Integration von weiteren Komponenten in ein vorhandenes System (wie Plug and Play). Durch das Public Subscribe Paradigma können OPC UA und MQTT einfach erweitert werden.

Konformität stellt alle vorher genannten Eigenschaften sicher. Es beschreibt, ob die Produkte vor dem Vertrieb geprüft werden bzw. ob es eine Zertifizierung gibt. Durch die OPC Foundation wird die Konformität von OPC und OPC UA gewährleistet.

Bekanntheitsgrad beschreibt den Verbreitungsgrad des Protokolls in der Industrie und die Verfügbarkeit von Komponenten. Eigentlich sind alle Technologien bekannt, MTConnect jedoch nur in der Fertigungstechnik und hier vor allem in den USA.

Verwendung in Unternehmen erläutert wie weit die Technologien aktuell in produzierenden Unternehmen eingesetzt werden. Bei diesem Punkt haben alle Technologien noch ein Verbesserungspotential. Von Unternehmen wird jedoch vermehrt OPC UA eingesetzt.

Die Auflistung in Tabelle 3.1 zeigt, dass OPC UA bei fast allen Kriterien Vorteile gegenüber den anderen Technologien aufweist. Lediglich die Verbreitung in Unternehmen ist noch nicht in vollem Umfang gegeben. Die Überlegenheit von OPC UA ist der Grund, warum diese Technologie sowohl für Standardisierungsbemühungen (siehe 2.2.4) als auch in dieser Arbeit eine große Rolle spielt. Durch die Vereinfachung der Erstellung von Informationsmodellen kann zur Verbreitung der Technologie beigetragen werden.

KAPITEL 4

Modellierung von CPPS: Ansätze, Modelle und Metamodelle

Modelle werden dazu verwendet, um einen Ausschnitt der Realität abzubilden [134]. Nach Broy zielt die Modellbildung der Informatik auf die Darstellung wesentlicher Strukturen, deren Zusammenhänge und Vorgänge unter gegeben Aufgabenstellungen und Gesichtspunkte [18].
Metamodelle hingegen beschreiben die abstrakte Syntax von Sprachen bzw. Modellen. Es sind somit die "Blaupausen", wie Modelle erstellt werden können.

Für die Modellierung können verschieden Ansätze verfolgt werden. In den letzten Jahren werden vermehrt gesamtheitliche Ansätze verfolgt, welche eine Unterstützung für die Erstellung der Modelle ermöglichen. Bekannte Vertreter sind Model Driven Architecture (MDA), Model Integrated Computing, generatives Programming [136].

4.1 Model Driven Engineering und Model Driven Architecture

Beim Model Driven Engineering (MDE) wird die Abstraktion durch Modelle dazu verwendet, die Komplexität eines Systems zu reduzieren [17, 129]. MDE folgt dem einfachen Prinzip: *Everything is a model*, um die Koherenz von modellgetriebenen Techniken sicherzustellen. Dies soll in derselben Art und Weise hilfreich sein, wie das Prinzip *Everything is an object* geholfen hat, objektorientierte Techniken zu vereinfachen und generalisieren [21]. Historisch wurde MDE hauptsächlich im Softwareengineering eingesetzt [129, 21], aber in den letzten Jahren findet MDE auch vermehrt Einsatz in industriellen Automatisierungsanwendungen [149]. Ein Kernpunkt von MDE ist es das Engineering mit formalen Modellen (Maschinen lesbar und ausführbare Repräsentationen) zu unterstützen. Auf dieser Basis bietet die Modellierung einige Vorteile, um den Engineering Prozess

effizienter und effektiver zu gestalten. Die Verwendung von Modellen erlaubt es die einzelnen Schritte des Engineering Prozesses, das sind (i) Validieren, (ii) Testen, (iii) Verifizieren, (iv) Simulieren, (v) Transformieren und (vi) Ausführen, zu automatisieren. Obendrein wird die Nachverfolgbarkeit von Engineering Artefakten verbessert, welches zu einer Verbesserung des Qualitätsmanagements führt.

2001 wurde Model Driven Architecture (MDA) von der Object Management Group (OMG)[1] als konkreter MDE Ansatz Modelle in der Softwareentwicklung zu verwenden definiert [141]. MDA verfolgt drei primäre Ziele: (i) Portabilität, (ii) Interoperabilität, (iii) Wiederverwendbarkeit (Reusability)

Diese Ziele werden durch die Architektur vorgeben, welche eine Trennung von plattformspezifischen von plattformunabhängigen Modellen fordert [58]. MDA spezifiziert drei Basismodelle entsprechend den drei Zielen. Diese Modelle spiegeln die unterschiedlichen Ebenen der Abstraktion wider, da in jeder Phase eine Anzahl von Modellen definiert werden kann. Jedes liefert eine detailierte Sicht auf das System (User Interface, Informationen, Engineering, Architektur, etc.) [86].

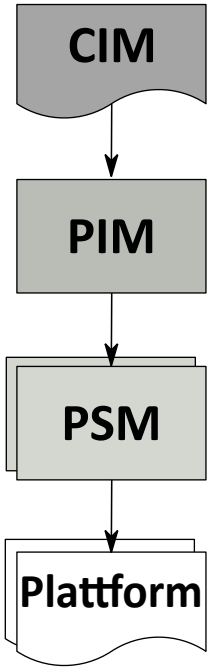

Abbildung 4.1: MDA Architektur (vgl. [17])

[1] http://www.omg.org/

UML ist die bevorzugte Modellierungssprache für MDA. Der wichtigste Vorteil der Verwendung von UML für Model-driven-Architecture ist, dass sie ein Format für den Informationsaustausch zwischen Tools und Anwendungen vorschlägt. Neben UML sind XML Metadata Interchange (XMI) [91] und die Meta-Object Facility (MOF) [89] die wichtigsten Standards im Zusammenhang mit MDA.
Abbildung 4.1 zeigt die grundsätzliche Vorgehensweise bei MDA. Der folgende Abschnitt gibt einen Auszug aus der Spezifikation [88] und beschreibt die wesentlichen Phasen:

CIM Computation Independent Model (CIM) ist ein Modell, welches einen fachlichen Entwurf mit einem Konzept oder Lastenheft darstellt. Dieses Modell wird vielfach auch als *Domain Model* oder *Business Model* bezeichnet, da es Vokabular und Begriffe von Domänenexperten verwendet [73]. Es zeigt im Groben die Funktion des Systems, ohne auf technologiespezifische Aspekte einzugehen [39]. Das Computation Independent Model (CIM) schließt somit die Lücke zwischen Domänenexperten und den Technikern, welche für eine Umsetzung des Systems verantwortlich sind [141].

PIM Das Platform Independent Model (PIM) ist das Kernmodell der MDA und bietet eine abstrakte Sicht auf das System, ohne dabei auf plattformspezifische Aspekte einzugehen. Üblicherweise wird hier versucht, mit Hilfe von UML-Profilen, welche das UML-Metamodell um Stereotypen erweitern, das komplette System zu beschreiben. Diese Modelle sollen später automatisiert in das Platform Specific Model (PSM) transformiert werden.

PSM Durch die Anreicherung des Platform Independent Model (PIM) mit Plattform spezifischen Informationen erhält man das Platform Specific Model (PSM). Hier wird das Modell dann an die entsprechenden Plattformen wie z.B. *.Net* angepasst. Aus dem Platform Specific Model (PSM) kann entweder ein lauffähiger Code generiert werden, oder das Modell ist bereits ausführbar (*Executable Model*).

Transformation Ein wesentlicher Bestandteil von MDA ist die (automatische) Transformation der Modelle. Erst dies ermöglicht es, den Softwareentwicklungsprozess zu automatisieren, da sie die essentiellen Mechanismen für die Manipulation von Modellen bereitstellen [17, 129]. Für die Transformation von Modellen kann man grundsätzlich zwischen zwei Vorgehensarten unterscheiden. Einerseits können Model-To-Text (M2T) Transformationen verwendet werden. Diese werden üblicherweise für die Erstellung von Softwareartefakten (z.B. SourceCode) verwendet. Anderseits gibt es die Model-To-Model Transformation (M2M), welche eine direkte Transformation von einem Modell in ein anderes erlaubt. Dabei wird im Normalfall das Modell für den jeweiligen Anwendungsfall spezialisiert. Dafür gibt es eine Unzahl von unterschiedlichen Sprachen, welche vom jeweiligen Anwendungsfall stark abhängig sind.

Bekannteste Sprachen sind QVT[2] Acceleo[3] ATL[4] VIATRA[5] DSLTrans[6] [22, 27]. Die Spezifikation für eine solche Transformation ist in Regeln (Mapping Rules) definiert [39].

4.2 ATLAS Transformation Language (ATL)

Atlas Transformation Language (ATL)ist eine Modelltransformationssprache und ein Toolkit. Im Bereich des MDE bietet ATL Möglichkeiten, aus einem Satz von Quellmodellen einen Satz von Zielmodellen zu erzeugen [4], [54]. Im Rahmen der ATL-Sprache wird die Generierung von Zielmodellelementen durch die Spezifikation von Transformationsregeln erreicht. Die ATL-Sprache basiert auf der OMG OCL-Norm (Object Constraint Language) [87] sowohl für ihre Datentypen als auch für ihre deklarativen Ausdrücke. Es gibt einige Unterschiede zwischen der OCL-Definition und der aktuellen ATL-Implementierung.

Anhand eines Beispiels werden die wesentlichen Bestandteile einer ATL Transformation erläutert. Abbildung 4.2

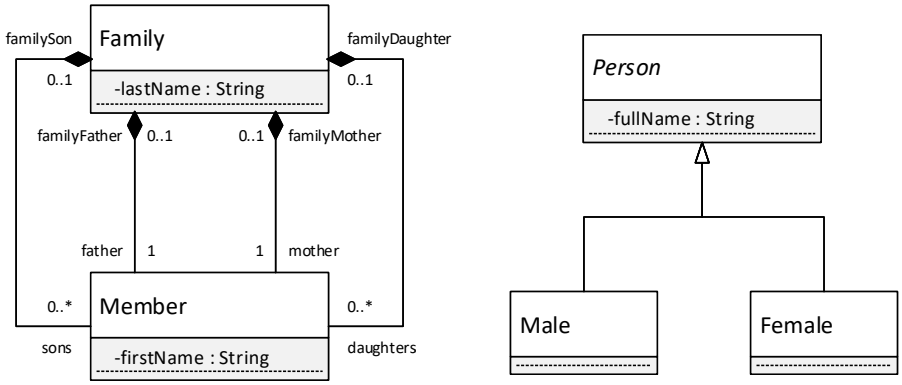

Abbildung 4.2: Modelle von Family und Person

Codeabschnitt 4.1 zeigt, wie die Klasse *Familie* in eine Klasse *Person* transformiert werden soll.

```
1 module Families2Persons;
2 create OUT : Persons from IN : Families;
3
4 helper context Families!Member def: familyName : String =
```

[2]http://www.omg.org/spec/QVT/
[3]http://www.eclipse.org/acceleo/
[4]http://www.eclipse.org/atl/
[5]http://eclipse.org/viatra/
[6]https://github.com/githubbrunob/DSLTransGIT

```
5     if not self.familyFather.oclIsUndefined() then
6        self.familyFather.lastName
7     else
8        if not self.familyMother.oclIsUndefined() then
9           self.familyMother.lastName
10       else
11          if not self.familySon.oclIsUndefined() then
12             self.familySon.lastName
13          else
14             self.familyDaughter.lastName
15          endif
16       endif
17    endif;
18
19 helper context Families!Member def: isFemale() : Boolean =
20    if not self.familyMother.oclIsUndefined() then
21       true
22    else
23       if not self.familyDaughter.oclIsUndefined() then
24          true
25       else
26          false
27       endif
28    endif;
29
30 rule Member2Male {
31    from
32       s : Families!Member (not s.isFemale())
33    to
34       t : Persons!Male (
35          fullName <- s.firstName + ' ' + s.familyName
36       )
37 }
38
39 rule Member2Female {
40    from
41       s : Families!Member (s.isFemale())
42    to
43       t : Persons!Female (
44          fullName <- s.firstName + ' ' + s.familyName
45       )
46 }
```

Codeabschnitt 4.1: ATL Beispiel[7]

Eine ATL Transformation besteht aus einem Satz aus *Rules* und *Helper*, welche im Anschluss an den Header aufgelistet werden. Die wichtigsten Elemente sind:

Header Der Header-Abschnitt benennt das Transformationsmodul und die Variablen, welche dem Quell-und Zielmodelle (*IN* und *OUT*) entsprechen, zusammen mit ihren Metamodellen (*Persons* und *Families*), die als Typen fungieren. Der Kopfbereich von *TypA2TypB* ist:

[7] http://www.eclipse.org/atl/documentation/basicExamples_Patterns/article.php?file=SideEffect/index.html

```
1 module Families2Persons;
2 create OUT : Persons from IN : Families;
```

Helper ATL Helper können als ATL Equivalent zu Methoden gesehen werden. Ein Helper ist eine Hilfsfunktion, die ein Ergebnis berechnet, das in einer Regel benötigt wird. Der Helper *isFemale* ermittelt das Geschlecht des aktuellen Mitglieds.

Rules ATL definiert verschiedene Arten von Transformationsregeln: die *Matched Rules*, *Lazy Rules* und die *Called Rules*. Eine *Matched Rule* ermöglicht es, einige Modellelemente eines Quellmodells abzugleichen und daraus eine Reihe von unterschiedlichen Zielmodellelementen zu generieren. Eine *Matched Rule* wird mit dem Wort *rule* begonnen. Im Beispel gibt es zwei Regeln, die Regeln *Member2Male* und *Member2Female*. *Lazy Rules* sind wie *Matched Rules*, werden aber nur angewendet, wenn sie von einer anderen Regel aufgerufen werden. ATL definiert eine zusätzliche Art von Regeln *Called Rules*, die es ermöglichen, aus imperativem Code explizit Zielmodellelemente zu generieren.

Eine vollständige Auflistung aller Elemente von ATL ist von der Eclipse Foundation verfügbar[8].

4.3 Unified Modelling Language - UML

DieUnified Modelling Language (UML) ist die verbreitetste Notationsmethode, um Softwaresysteme zu analysieren und zu entwerfen [125]. Mitte der 1990er wurde von Rumbaugh, Booch und Jacobsen der Versuch gestartet, eine einheitliche Methode für die Analyse von Systemen zu entwickeln. Da die Analyse jedoch kein einheitlicher, sondern ein kreativer Prozess ist, konnte keine einheitliche Methodik entwickelt werden. Jedoch wurde eine einheitliche Notationsmethode entwickelt, mit der alle Phasen der Analyse abgedeckt werden konnten. Dies war die Geburtstunde von UML (Unified Modeling Language), welche schließlich auch als IEC 19501 [44] standardisiert wurde. Die ursprüngliche Version von UML vereinte die Methoden von Booch [15], Rumbaugh [122] und Jacobson [52]. Im Laufend wurde UML weiterentwickelt und Ideen und Konzepte anderer ebenfalls in den Standard integriert. In der aktuellen Version hat UML nur mehr wenig mit den anfänglichen Versionen gemein. Die aktuelle Spezifikation von UML definiert 7 Strukturdiagramme und 7 Verhaltensdiagramme. Die wichtigsten Diagramme, welche auch im Zuge der Arbeit verwendet werden, sind:

- Klassendiagramm
- Anwendungsfalldiagramm
- Aktivitätsdiagramm

[8] https://wiki.eclipse.org/ATL/User_Guide_-_Overview_of_the_ATL_Language

- Zustandsdiagramm

Mit UML Profilen lassen sich die UML Modelle anpassen, damit sie noch besser ein Problem beschreiben können. UML Profile stellen eine leichten Erweiterungsmechanismus für die UML dar. Sie basieren auf benutzerdefinierten Stereotypen und Tagged Values, die auf Elemente, Attribute, Methoden, Verbindungen und Verbindungsenden angewendet werden können. Profile sind deshalb leichtgewichtig, weil sie es nicht erlauben bestehende Metamodelle zu modifizieren oder ein neues Metamodell zu erstellen, wie es z.B. MOF [89] tut. Das Profil erlaubt nur die Anpassung oder Anpassung eines bestehenden Metamodells mit Konstrukten, die für eine bestimmte Domäne, Plattform oder Methode spezifisch sind. Es ist nicht möglich, Einschränkungen, die für ein Metamodell gelten, zu entfernen, aber es ist möglich, neue Einschränkungen hinzuzufügen, die spezifisch für das Profil sind [131].

4.3.1 Klassendiagramm (Class Diagram)

Abbildung 4.3 zeigt ein Beispiel für ein Klassendiagramm und Abbildung 4.4 das vereinfachte Metamodell von UML Klassendiagrammen. Die vollständige Spezifikation von UML Klassendiagrammen kann aus der OMG Unified Modeling Language Spezifikation entnommen werden [90].

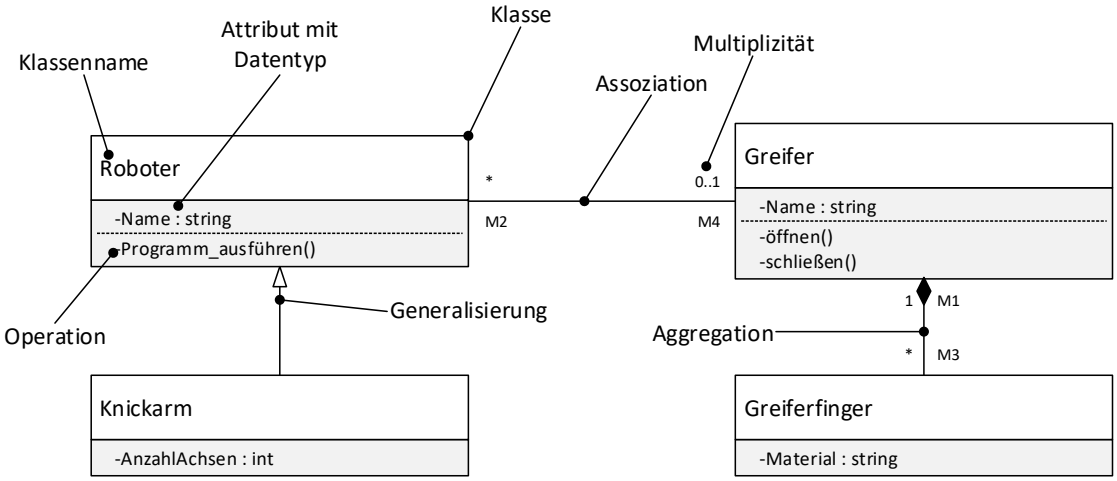

Abbildung 4.3: Notation UML Klassendiagramm

Das Klassendiagramm stellt das wichtigste Diagramm in UML dar. Klassendiagramme können in fast allen Phasen eines Projektes eingesetzt werden. In der Analysephase als Modell der Wirklichkeit, in der Designphase kann damit komplexe Software modelliert werden und in der Implementierungsphase sogar als Quelle für den Sourcecode dienen.

Das Klassendiagramm basiert auf objektorientierten Prinzipien wie Abstraktion, Kapselung

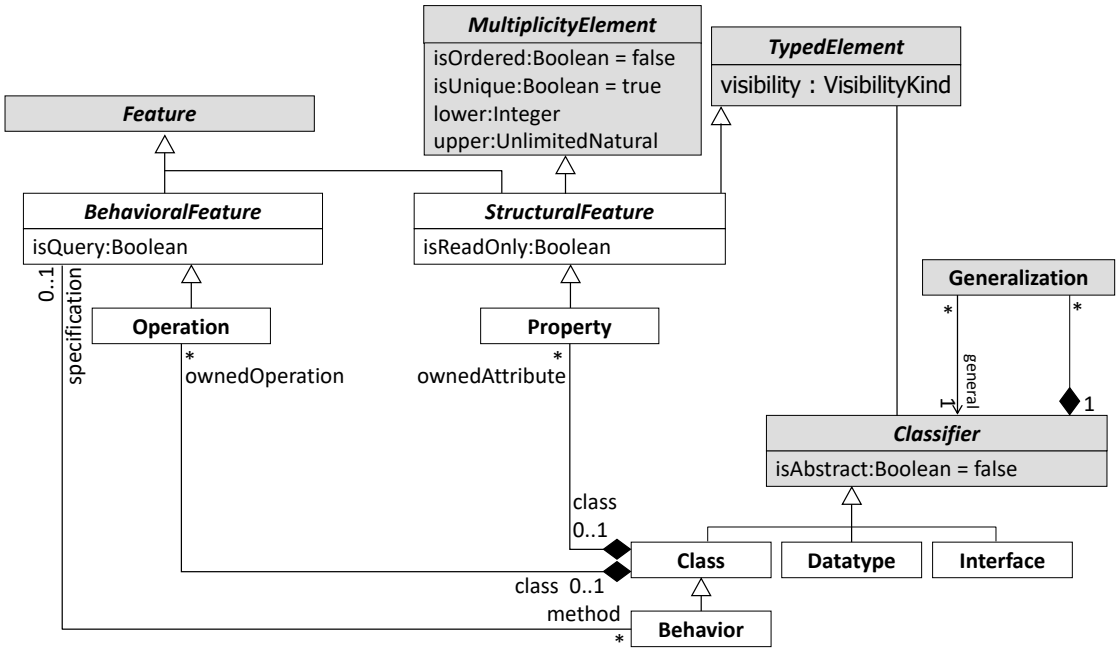

Abbildung 4.4: Vereinfachtes Metamodell für UML Klassendiagramme

und Vererbung. In Klassendiagrammen werden Klassen und deren Beziehungen untereinander dargestellt. Dabei gibt es unterschiedliche Arten von Beziehungen, die Assoziation, die Aggregation bzw. Komposition und die Generalisierung bzw. Spezialisierung. Jede Klasse verfügt über Methoden und Attribute, mit denen die Struktur und das Verhalten von Klassen beschrieben werden. Neben einfachen Klassen bieten UML Klassendiagramme noch Interfaceklassen und parametrierbare Klassen.

Eine Übersicht der wichtigsten Elemente eines UML Klassendiagramms wird im Folgenden gegeben:

Klasse (Class) Eine Klasse beschreibt eine Menge von Instanzen, welche dieselben Merkmale und Semantik besitzen. Klassen werden durch Rechtecke dargestellt, die in drei Bereichen (Compartments) - Klassenname, Attribute, Operation gegliedert sind. Dabei ist der Klassenname erforderlich, Attribute und Operationen sind optional. Klassennamen beginnen üblicherweise mit einem Großbuchstaben und sind meist Substantive im Singular. Die Attribute einer Klasse werden mit ihrem Namen angegeben und können zusätzlich Angaben (Typ, Initialwert, Eigenschaftswerte) enthalten. Für Methoden gilt diese Regel ebenfalls. Ist der Klassenname kursiv, so handelt es sich um eine abstrakte Klasse, von der niemals Instanzen erzeugt werden können. Diese bilden die Basis für Unterklassen, von denen Instanzen erstellt werden können.

Attribut (Property) Ein Attribut ist ein (Daten-)Element, das in jeder Instanz einer

Klasse enthalten ist und von den Instanzen(Objekten) mit einem individuellen Wert dargestellt wird. Jedes Attribut wird durch seinen Namen beschrieben, der üblicherweise mit einem kleinen Buchstaben beginnt. Zusätzlich können der Typ, die Sichtbarkeit und ein Initialwert definiert werden. Die vollständige Syntax lautet:

```
1       [Sichtbarkeit][/]Name[:Typ][Multiplizitaet][=Initialwert]
```

Alle Klassen haben je nach Zugriffsmodifikator (Sichtbarkeit) unterschiedliche Zugriffsebenen. Hier sind die Zugriffsebenen mit den entsprechenden Symbolen: Public (+), Private (-), Protected (#), Package ( ), Derived (/), Static (underlined). Die Multiplizität (Kardinalität) beschreibt die Menge der möglicher Ausprägungen. Die Multiplizität wird durch eine obere und untere Schranke beschrieben, z. B. 3. . . 7. Entspricht die obere Schranke der unteren, so kann die Unter- oder Obergrenze entfallen. Ist die untere Grenze 0, so bedeutet dies, die Beziehung ist optional.

Methoden (Operationen) Methoden werden dazu verwendet, damit Klassen anderen Klassen Funktionalität zur Verfügung stellen können. Über Methodenaufrufe kommunizieren die aus den Klassen instanziierten Objekte miteinander und können ihr Verhalten koordinieren.

Beziehungen Es gibt vier verschiedene Arten von Beziehungen zwischen Klassen. Diese lauten:

Assoziation Die Assoziation beschreibt die Kommunikation zwischen zwei Klassen. Diese wird mit einer einfachen Linie dargestellt. Durch einen Pfeil kann man eine gerichtete Assoziation modellieren.

Aggregation Eine Aggregation ist eine spezielle Assoziation. Es stellt dar, dass eine Klasse eine Teil von einer anderen ist. Eine Aggregation beschreibt, wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt. Die Notation der Aggregation ist eine Line mit einer Raute auf der Seite des Ganzen.

Komposition Eine Komposition ist eine strenge Form der Aggregation, bei der die Teile vom Ganzen existenzabhängig sind, d.h. die einzelnen Teile können ohne das Gesamte nicht existieren. Die Komposition besitzt dieselbe Notation wie die Aggregation, jedoch wird die Raute ausgefüllt dargestellt.

Generalisierung Eine Generalisierung ist eine Beziehung zwischen einer allgemeinen und einer speziellen Klasse, wobei bei der Spezialisierung weitere Merkmale zu jenen der allgemeinen Klasse hinzugefügt werden können.
Bei der Generalisierung erfolgt eine hierarchische Gliederung der Merkmale. Merkmale von allgemeinerer Bedeutung werden allgemeineren Klassen (Superklassen) zugeordnet und speziellere Merkmale werden Subklassen zugeordnet, die den allgemeineren Klassen untergeordnet sind. Merkmale der Superklasse werden an die Subklassen vererbt. Spezialisierungen erben alle Merkmale ihrer Superklassen, können diese jedoch ergänzen bzw. überschreiben. Mit dieser Beziehung lassen sich hierarchische Vererbungsbäume darstellen, welche wichtige Elemente bei der Modellierung von Softwarearchitekturen sind.

4.3.2 Anwendungsfalldiagramm (Use Case Diagram)

Use Case Diagramme können dazu verwendet werden einen guten Überblick über ein System auf einem hohen abstrakten Niveau (Abstraktionsniveau) zu erlangen. Das Diagramm beschreibt, wie User oder Systeme (Akteure) mit einem System interagieren können. Grundsätzlich beschreibt ein Use Case Diagramm nicht das Verhalten und die Abläufe von Systemen, sondern nur die Zusammenhänge zwischen einer Menge von Anwendungsfällen und der daran beteiligten Akteure.
UML Use Case-Diagramme sind ideal für: (i) Darstellung der Ziele von System-Benutzer-Interaktionen, (ii) Definition und Organisation der funktionalen Anforderungen in einem System, (iii) Festlegung von Kontext und Anforderungen an ein System, (iv) Modellierung des grundlegenden Ablaufs von Ereignissen in einem Anwendungsfall

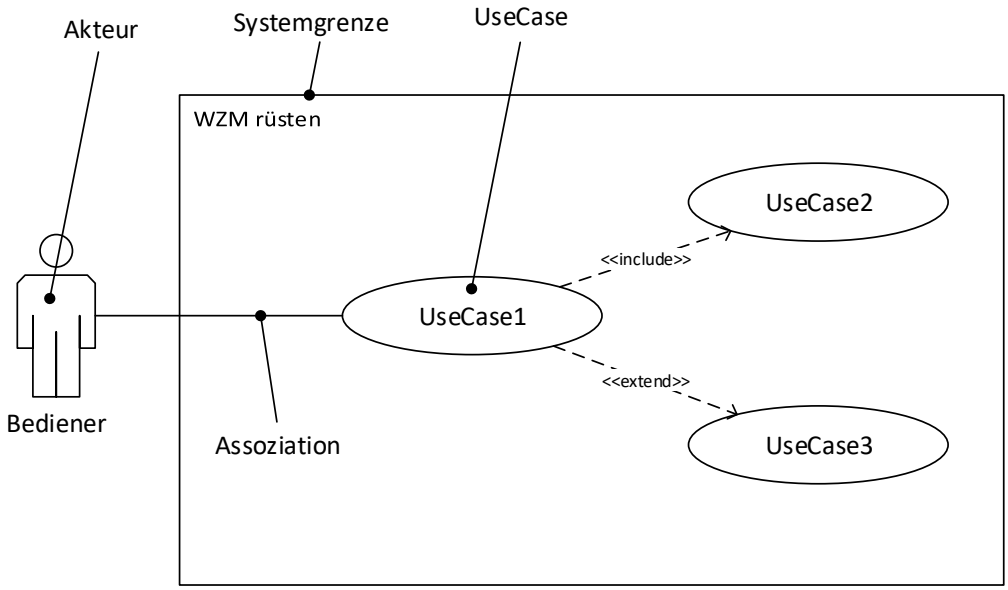

Abbildung 4.5: Beispiel UML Use Case Diagramm

Use-Case Diagramme verwenden folgende Elemente (siehe Abbildung 4.5):

Akteure werden als Strichmännchen dargestellt, welche sowohl Personen wie Kunden oder Administratoren als auch ein System darstellen können (bei Systemen wird manchmal auch ein Bandsymbol verwendet).

Anwendungsfälle werden in Ellipsen dargestellt. Sie müssen beschrieben werden (z. B. in einem Kommentar oder einer eigenen Datei).

Assoziationen zwischen Akteuren und Anwendungsfällen müssen durch Linien gekennzeichnet werden.

Systemgrenzen werden durch ein Rechteck gekennzeichnet.

include-Beziehungen werden mittels (mit «include» gekennzeichneter) gestrichelter Linie und einem Pfeil zum inkludierten Anwendungsfall gekennzeichnet, wobei dieser für den aufrufenden Anwendungsfall notwendig ist.

extend-Beziehungen werden mittels (mit «extend» gekennzeichneter) gestrichelter Linie und einem Pfeil vom erweiternden Anwendungsfall gekennzeichnet, wobei dieser von dem aufrufenden Anwendungsfall aktiviert werden kann, aber nicht muss.

4.3.3 UML Zustandsdiagramm (State Machine Diagramm)

Zustandsdiagramme gehen auf David Harel zurück [40] und wurden danach in die UML aufgenommen.

Ein Zustandsdiagramm beschreibt die Abfolge von Zuständen, welches ein Objekt während seines Lebenszyklus annehmen kann. Zustandsdiagramme beschreiben außerdem die Operationen bzw. die Ereignisse, die zu einer Zustandsänderung führen [56].

Grundsätzlich kann jedes System mit einem Zustandsdiagramm modelliert werden für das folgende Regeln gelten:

- Das Objekt befindet sich immer in einem definierten Zustand
- Nie zugleich in mehreren Zuständen
- ein Zustand kann auch Subzustände aufweisen
- Jedes Zustandsdiagramm muss einen Anfangszustand aufweisen und kann optional über einen Endzustand verfügen
- Übergänge zwischen Zuständen werden immer von einem Ereignis ausgelöst

Abbildung 4.6 zeigt ein Beispiel eines UML Zustandsdiagrammes, während Abbildung 4.7 das dazugehörige Metamodell mit den wichtigsten Elementen darstellt. Nachfolgend werden die wichtigsten Notationselemente dieses Diagrammtyps erklärt [56], [90], [125].

Zustände (States) Zustände werden durch abgerundete Rechtecke modelliert. Sie können einen Namen beinhalten und optional durch horizontale Linien in bis zu drei Bereiche eingeteilt werden. Im oberen Bereich (Abbildung 4.6) steht üblicherweise die Bezeichnung des Zustandes. In einem weiteren Bereich können existierende Zustandsvariablen mit typischen Wertebelegung angeführt werden.
Im unteren Bereich kann angeführt werden, welche internen Verhaltensweisen in diesem Zustand ausgeführt werden. Diese Verhaltensweisen sind in UML mit ihren entsprechenden Auslösern definiert. Diese sind Schlüsselwörter und dürfen nur als solche verwendet werden. Das Schlüsselwort „entry“ wird verwendet, wenn ein Verhalten des Systems beim Eintritt in den entsprechenden Zustand ausgeführt

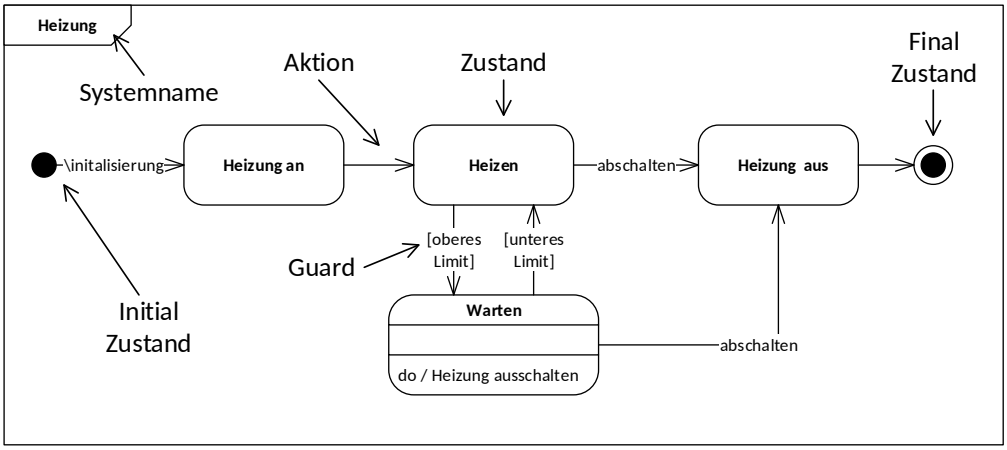

Abbildung 4.6: Notation UML Zustandsdiagramm

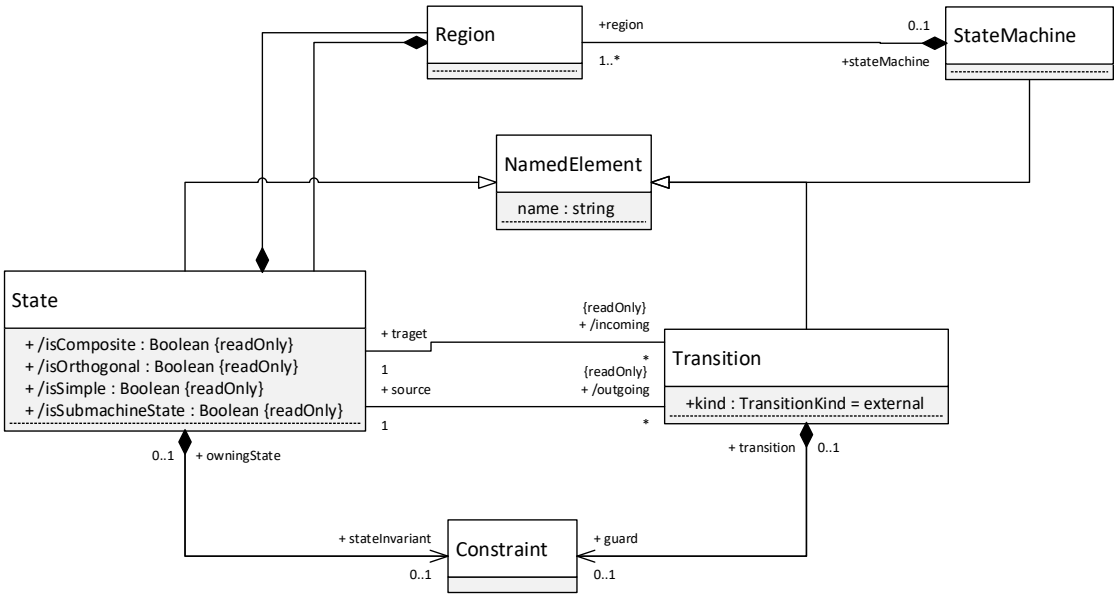

Abbildung 4.7: Metamodell UML State Machines

werden soll. Mit dem Wort „do“ wird das Verhalten definiert, das während dem Verharren in dem Zustand ausgeführt werden soll und der Auslöser „exit“ beschreibt das Verhalten beim Verlassen des Zustandes.

Transition Die Transition beschreibt die Ereignisse oder Bedingungen, die den Wechsel von einem Zustand zum nächsten hervorrufen. Sie bilden den Übergang von einem Ausgangs- zu einem Zielzustand.
Der Trigger ist der Auslöser für die Transition. Die Angabe des Triggers ist aber

optional. Der Guard repräsentiert eine Bedingung, die wahr sein muss, damit die Transition ausgeführt werden kann. Die Bedingungen werden stets in eckigen Klammern angegeben. Danach wird das Verhalten beim Durchlaufen der Transition angegeben. Auch die Angabe des Guards und des Verhaltens ist optional.

Startzustand Der Startzustand bildet den Startpunkt des Zustandsdiagrammes, von dem aus der erste Zustand des Systems erreicht wird. Von dem Startpunkt darf nur eine Transition ausgehen und diese darf keine Trigger und Guards beinhalten, damit gewährleistet wird, dass der erste Übergang sicher durchlaufen wird und eindeutig ein Zustand erreicht wird.

4.4 OPC Unified Architecture (OPC UA)

Die Basiskonzepte der Modellierung in OPC UA sind Knoten (*Nodes*) und Referenzen (*References*). Ein Node entspricht in etwa einem Objekt in der objekt-orientierten Programmierung. Jeder Node besitzt einige Attribute, welche gelesen und beschrieben werden können. References verbinden Nodes miteinander.
In OPC UA wird zwischen dem Adressraummodell (*AddressSpaceModel*), dem Informationsmodell (*InformationModel*) und dem Adressraum (*AddressSpace*) unterschieden. Abbildung 4.8 zeigt den Zusammenhang zwischen Adressraummodell, Informationsmodell und realen Daten. Das Adressraummodell spezifiziert die Bausteine, mit denen Informationsmodelle gebildet werden können, welche wiederum für den Adressraum eines OPC UA Servers benötigt werden. Schlussendlich werden die konkreten Daten eines Servers basierend auf dem Informationsmodell gebildet

4.4.1 Adressraummodell (AddressSpace Model)

Das Adressraummodell wird aus verschiedenen *NodeClasses* mit Attributen, sowie einigen Standard Nodes gebildet (z.B. Referenzen wie *HasSubtype*) [72]. Abbildung 4.9 stellt eine vereinfachte Version des OPC UA Metamodells dar. Das vollständige Metamodell ist in der OPC UA Spezifikation Teil 3 zu finden [100]. Die *NodeClasses*, aus welchen das Adressraummodell gebildet wird, sind in Abbildung 4.9 dargestellt. Nachfolgend erfolgt eine Beschreibung der wesentlichen *NodeClasses*.

Base Alle *NodeClasses* sind Spezialisierungen der Klasse *Base*. Das Metamodell der *NodeClasses* ist in Abbildung 4.9 dargestellt. Jedes Element in OPC UA verfügt über die Attribute *NodeId*, *BrowseName*, *NodeClass*, *DisplayName*, *Description*, *WriteMask* und *UserWriteMask*. Einige davon sind jedoch optional.

Der *BrowseName* Attribut wird verwendet, um einem Knoten einen für den Menschen verständlichen Namen beim Durchsuchen des Adressraumes zur Verfügung zu stellen. Es ist kein eindeutiger Name, da mehrere Knoten im Adressraum den selben *BrowseName* haben können. Der *DisplayName* hingegen verfügt über eine Lokalisierung, d.h., der Name kann in unterschiedlichen Sprachen angezeigt werden. Er wird dazu verwendet, um den Namen eines Knotens über einen Client ausgeben

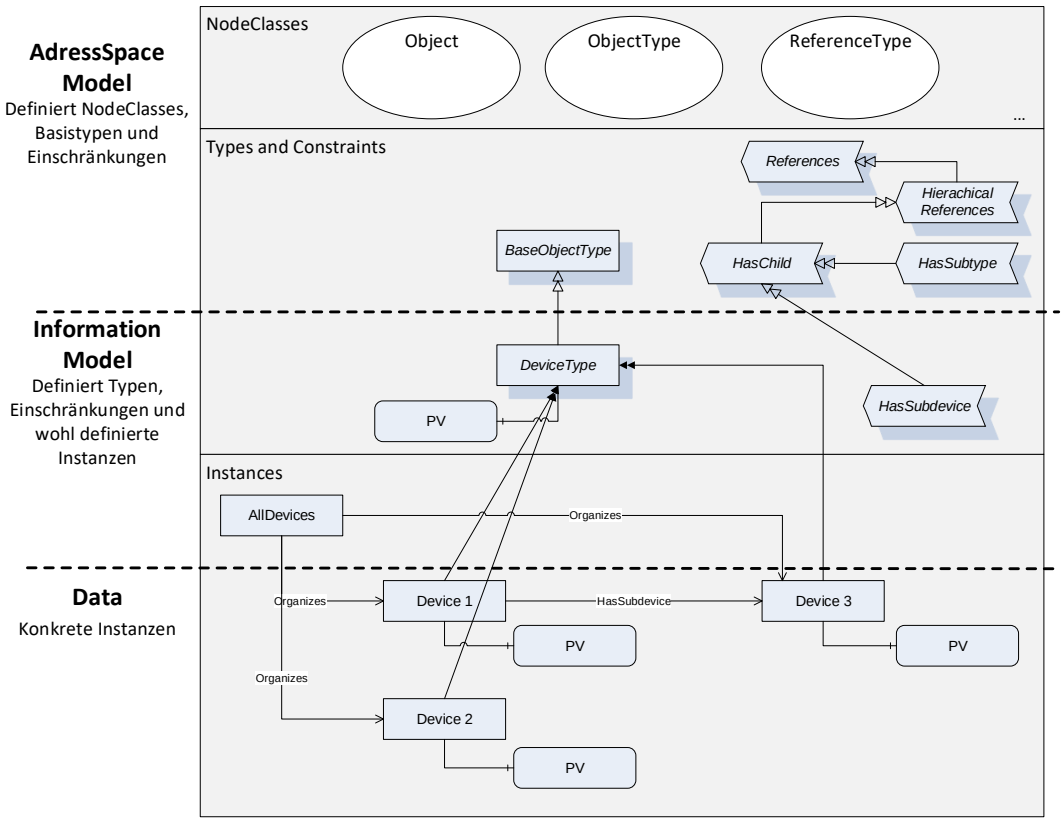

Abbildung 4.8: Zusammenhang Adressraummodell, Informationsmodell und Daten [72]

zu lassen. Die *Description* soll den Knoten näher beschreiben. Sie ist wie der *DisplayName* vom Typ *LoacalizedText*. *WriteMask* und *UserWriteMask* geben die Möglichkeit vor, wie ein Client die Attribute eines Knotens beschreiben kann. Sie sind ein 32 Bit unsigned Integer Strukt, welches in der OPC UA Spezifikation Teil 3 beschrieben ist. Der Unterschied zwischen den beiden Attributen liegt darin, dass bei der *UserWriteMask* die User Zugriffsrechte von OPC UA berücksichtigt werden.

ObjectType Mit den *ObjectTypes* können Blaupausen für *Objects* erstellt werden. Von diesen Typen können dann Objekte abgeleitet werden. Die *ObjectType* NodeClass besitzt zusätzlich das Attribut *IsAbstract*. Ist der ObjectType abstrakt, so können keine Objects davon instanziiert werden, sondern nur von den Subtypen. Einer der wichtigsten Objekttypen ist der *FolderType*, welcher zum Strukturieren des Addressraumes verwendet wird.

Object *Objects* werden dazu verwendet, um Systeme bzw. Objekte aus der realen Welt sowie Software-Objekte zu beschreiben. Zusätzlich besitzt diese NodeClass das Attribut *EventNotifier*, welches angibt, ob sich die Node Events abonnieren können.

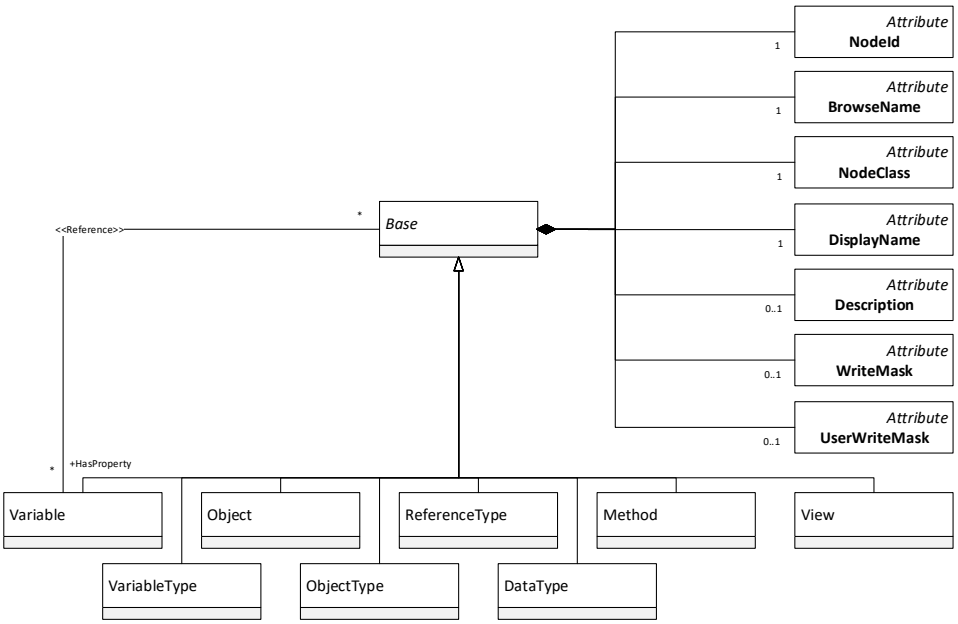

Abbildung 4.9: OPC Base NodeClass [100]

VariableType VariableTypes bieten Typdefinitionen für Variables. Die VariableType NodeClass besitzt zusätzlich die Attribute *Value*, *DataType*, *ValueRank*, *ArrayDimensions* und *IsAbstract*.

Variable Die Variable NodeClass wird zum Repräsentieren von einfachen oder komplexen Werten verwendet. Bei Variables wird in OPC UA zwischen *Property* und *DataVariable* unterschieden. Properties sind Eigenschaften von *Objects*, *DataVariables* und anderen Nodes. DataVariables beschreiben hingegen den Inhalt von Objekten. Obwohl *Properties* und *DataVariables* von der gleichen NodeClass sind, unterscheiden sie sich durch ihre Attribute und Referenzen. Properties sind üblicherweise die "Blätter" in einer Node-Hierarchie und sollten immer im Kontext einer anderen Node definiert werden. DataVariables hingegen sind für die Modellierung von komplexen Strukturen vorgesehen.

Methods Methods definieren aufrufbare Funktionen, die einem Object bzw. Object Type zugeordnet sind. Über einen Service Call können Methoden ausgeführt werden. Zusätzlich zu den Attributen, welche die Methoden von der Base Klasse erben, besitzen Methoden die beiden Attribute *Executable* und *UserExecutable*. Executable gibt an, ob die Methode gerade ausführbar ist. Bei UserExecutable werden zusätzlich die Anwenderrechte in Betracht gezogen.

DataType Die NodeClass *DataType* beschreibt die Syntax eines Datentyps, eines Werts einer *Variable* bzw. eines *VariableType*. Die DataTypes können entweder einfach

oder komplex sein. Die Zuweisung eines Datentyps zu einer Variable bzw. VariableType erfolg mit der *HasDataType* Referenz, welche auf ein Object der NodeClass DataType zeigt.

ReferenceType References sind Instanzen vom *ReferenceType*, welcher durch die entsprechende NodeClass definiert wird. Die *ReferenceTypes* werden im Adressraum dargestellt, während eine Reference Bestandteil einer Node ist [104]. Die OPC UA Spezifikation definiert eine Menge von vordefinierten *ReferenceTypes*, welche jedoch jederzeit ergänzt werden können. Der *BrowseName* der Reference liefert die Bedeutung der Referenz aus Sicht des Quellknotens. Ist der *BrowseName* zum Beispiel *HasComponent*, so sagt die Reference aus, dass der Quellknoten den Zielknoten als Komponente besitzt. Der *ReferenceType* besitzt, zusätzlich die Attribute *IsAbstract*, *Symmetric* und *InverseName*. Symmetric gibt an, ob die Bedeutung der *Reference* für Ursprungs- und Zielknoten gleich ist. Der *InverseName* beschreibt den Namen, wenn die Referenz vom Zielknoten zum Ursprungsknoten gelesen wird. *IsAbstract* gibt an, ob der *ReferenceType* instanziiert werden kann.

4.4.2 Informationsmodell (Information Model)

Das Informationsmodell verwendet das Adressraummodell, um domänenspezifische Typen und Instanzen zu definieren. Anschließend werden die Informationsmodelle als Basis für die Daten (den Adressraum) in einem OPC UA Server verwendet. Im Adressraum sind nicht nur die Typen (Typenraum), sondern auch deren Instanzen (Instanzraum) enthalten [97]. Wie in Kapitel 3.1 erwähnt, definiert OPC UA Basisinformationsmodelle für allgemein gültige Information (z.B. Alarme) in Teil 5 der Spezifikation. Sie beschreiben daher den Adressraum eines ”leeren” OPC UA Servers [102]. Das Basis Informationsmodell beschreibt laut Mahnke [72]:

- Einen Einstiegspunkt in den Adressraum. Dieser wird von den Clients benötigt, um sich durch die Instanzen und Typen eines OPC UA Servers navigieren zu können.
- Basis Typen, als Ausgangspunkte für verschiedene Typ-Hierarchien z.B. *BaseObjectType* und *BaseVariableType*.
- Vordefinierte aber erweiterbare *ObjectTypes* und *DataTypes*.
- Ein Server Object (*Server*), das Kapazitäts- und Diagnoseinformationen des Servers enthält.

Die Basis-Informationsmodelle in OPC UA sind in der folgenden Auflistung dargestellt. Zu den Basismodellen wurden auch weitere Informationsmodelle spezifiziert. Diese wurden im Anhang zur Spezifikation Teil 5 hinzugefügt. Hier sind die Modelle für *StateMachines* und *FileTransfer* als wichtigste Ergänzungen anzuführen.

Data Acess - DA Data Access (OPC UA Spezifikation Teil 8 [107]) ist ein Informationsmodell für Echtzeitdaten, d.h. Daten, die den aktuellen Zustand und das Verhalten

des zugrunde liegenden Industrie- oder Geschäftsprozesses darstellen. Der Zugriff erfolgt über Read, Write und Subscription Services. Typische Quellen sind Sensoren, Steuergeräte, Positionsgeber und mehr. Solche Informationen werden typischerweise von Clients für Benutzeranzeigen oder zur Überwachung und Steuerung des Prozesses verwendet. Das DA-Modell definiert auch Codes, welche die Qualität der physikalischen Verbindung festlegen.

Alarms & Conditions - AC Dieses Informationsmodell (OPC UA Spezifikation Teil 9 [108]) definiert, wie Zustände (Dialoge, Alarme) gehandhabt werden. Eine Zustandsänderung löst ein Event aus. Clients können sich für solche Events anmelden und auswählen, welche Werte sie als Teil des Eventreports erhalten wollen (z.B. Meldungstext, Quittierverhalten).

Historical Data Acess - HA Historical Data Access befasst sich mit dem Umgang mit Zeitreihendaten. Der AddressSpace von HDA-Servern enthält historische Knoten, die die Geschichte von Variablen und Eigenschaften darstellen. HDA-Clients arbeiten mit historischen Daten, indem sie über die HistoryRead- und HistoryUpdate-Services auf diese Knoten zugreifen. Vielfältige Aggregatfunktionen erlauben eine Vorverarbeitung im Server. Historischer Zugriff ist in OPC UA Spezifikation Teil 11 [106], Aggregate in Teil 13 [99] spezifiziert.

Programs OPC UA Programme stellen lang laufende, oft komplexe und zustandsbehaftete Funktionen in einem Server oder dem zugrundeliegenden System dar. Beispiele sind die Ausführung und Steuerung eines Batch-Prozesses oder die Ausführung eines Werkzeugmaschinenteilprogramms Jedes Programm stellt sich durch einen Zustandsautomaten dar, der Meldungen während Zustandsübergängen an den Client sendet. Die Ausführungszeit eines Programms ist nicht an die Lebensdauer einer OPC UA Sitzung gebunden. Das *Programs* Information Model ist spezifiziert in OPC UA Spezifikation Teil 10 [105].

State Machines Das State Machine Informationsmodell (OPC UA Spezifikation Teil 5 Annex B [102]) stellt Möglichkeiten zur Verfügung, um das diskrete Verhalten von Objekten zu modellieren. Dies erfolgt mit Hilfe von Zuständen (States) und Übergängen zwischen diesen (Transitions). State Machines werden als komplexe Objekte modelliert. Dabei werden spezielle ObjectTypes, VariableTypes und ReferenceTypes eingeführt, welche nach definierten Regeln, laut Spezifikation, eingesetzt werden müssen.

4.4.3 OPC UA Companion Specifications

Im nachfolgenden Abschnitt werden die wichtigsten herstellerspezifischen Erweiterungen von OPC UA erklärt.

OPC UA for Devices (IEC 62541-100) und Analyzer Devices OPC UA for Devices (DI) beschreibt ein Informationsmodell für Basistypen von konfigurierbaren

Komponenten und Geräten (Devices). Das Modell bietet außerdem Konzepte für das logische Gruppieren von Parametern, Methoden und Komponenten [93]. OPC Unified Architecture for Analyzer Devices (ADI) definiert ein Informationsmodell auf Basis des DI-Modells für komplexe Geräte [98].

OPC UA for Programmable Controllers based on IEC61131-3 OPC UA for IEC 61131-3 PLCOpen beschreibt die Implementierung des Softwaremodells nach IEC 61131-3 auf einem OPC UA Server Adressraum [111].

OPC UA for Field Device Integration OPC UA for Field Device Integration (FDI) definiert, wie ein Feldgerät über ein sogenanntes *Device Package* beschrieben werden muss. Dieses bestehen aus einer allgemeinen Parameterbeschreibung und Bedienelementen [32].

OPC UA for Enterprise and Control Systems based on ISA 95 OPC UA for ISA-95 Common Object Model definiert ein Informationsmodell für Produktionssteuerungssysteme für Chargen und MES [94].

MTConnect-OPC UA Companion Specification MTConnect-OPC UA garantiert die Interoperabilität und Konsistenz zwischen der MTConnect und OPC UA Spezifikation sowie Geräten, Software und anderen Produkten, welche einen dieser Standards implementieren [82].

OPC Unified Architecture for CNC Systems Diese Spezifikation wurde von einer gemeinsamen Arbeitsgruppe der OPC Foundation und des VDW erstellt. Es definiert ein OPC UA Informationsmodell für die Schnittstelle und den Datenaustausch mit Computerized Numerical Control (CNC) Systemen.
Ziel ist die Innovations- und Wettbewerbsfähigkeit von Werkzeugmaschinenherstellern und Herstellern von CNC-Systemen zu erhöhen, indem mit dieser Spezifikation die Kosten für die Verknüpfung von CNC-Systemen mit anderen Anwendungen gesenkt werden. Dies soll durch Standardisierung der CNC-Schnittstellen erreicht werden [146].

OPC Unified Architecture for AutomationML Die Idee für diesen Standard ist, dass Daten von Programmen korrekt und verlustfrei auf effiziente Weise exportiert und importiert werden können. Um Interoperabilität und eine nahtlose semantische Integration zu erreichen, muss dies durch einen effizienten Im- und Export der Daten ergänzt werden. OPC UA als ein bestehender und bereits vorhandener Kommunikationsstandard in der Automatisierungsebene (z.B. vorhandene Schnittstellen zu SPS, CNC und Softwaretools) bietet diese Möglichkeit. Ziel dieses Standards ist es, die Kommunikation und Operationalisierung von AutomationML mittels OPC UA und somit einen verlustfreien Austausch von OPC UA Systemkonfiguration mit AutomationML-Modellen zu gewährleisten [95].

4.4.4 Notation

Zur Modellierung des *AddressSpace* stellt OPC UA eine graphische Notation bereit. In Anhang D von Teil 3 der Spezifikation wird die Notation von OPC UA erklärt [100].

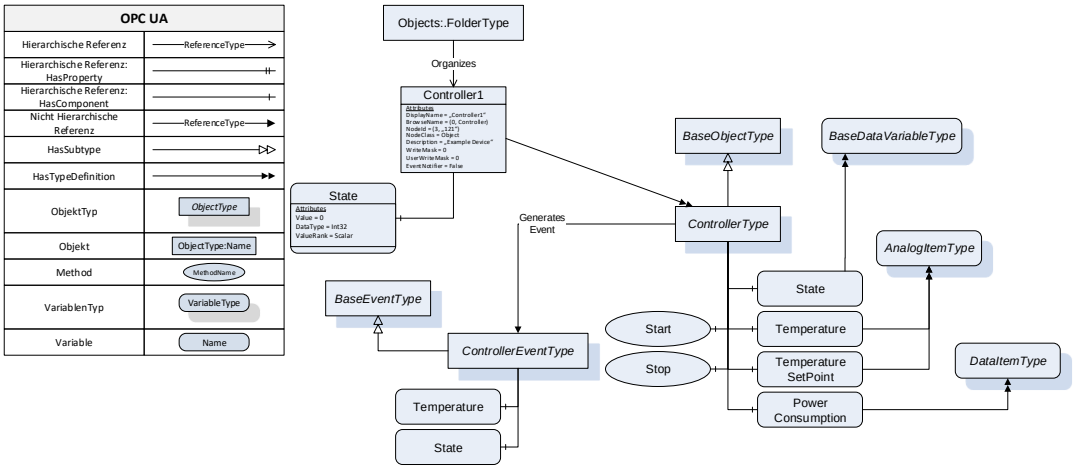

Abbildung 4.10: OPC UA graphische Notation [72], [100]

Die Notation erlaubt eine strukturelle Sicht auf OPC UA Nodes, deren Attribute und aktuellen Werte. Zusätzlich können Referenzen zwischen Nodes und deren Typen dargestellt werden. Aktuell gibt es jedoch keine Möglichkeit Events und historische Daten zu modellieren. Abbildung 4.10 zeigt eine Übersicht über die graphischen Elemente, welche für die Modellierung verwendet werden können.
In OPC UA gibt es zwei Formen der Darstellung: die einfache Notation und die erweiterte Darstellung, in der neben der Struktur auch Attribute und deren Werte dargestellt werden können. Zusätzlich kann auch noch der Typ des Knotens in der Form *«Name»::«Typ»* angegeben werden. Um alle notwendigen Informationen darzustellen, können einfache und erweiterte Darstellungsformen kombiniert werden.

4.5 MTConnect

Teil 2 des MTConnect Standards definiert das Informationsmodell für eine Vielzahl von Geräten (*Devices*). Das Informationsmodell basiert auf XML Schemata (XSD) und beschreibt die Daten, welche dem Agent zur Verfügung gestellt werden sollen. Diese XSD können von Herstellern modifiziert und wenn notwendig erweitert werden. Das Informationsmodell ist ein XMl Datenmodell, welches prinzipiell zwei unterschiedliche Elementtypen besitzt *Structural Elements* und *Data Elements* [77]. Codeabschnitt 4.2 zeigt ein solches XML Modell, welches aus verschachtelten XML Tags aufgebaut ist.

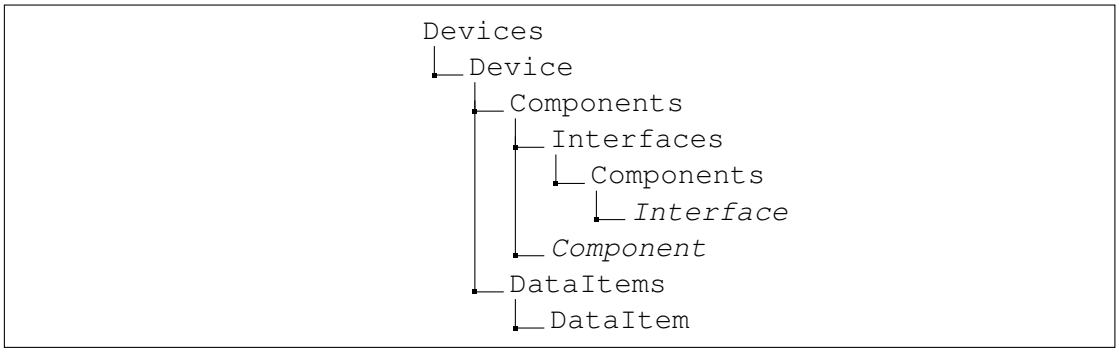

Abbildung 4.11: MTConnect Datenstruktur

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <MTConnectDevices xmlns:m="urn:mtconnect.org:MTConnectDevices:1.3" xmlns="urn
    :mtconnect.org:MTConnectDevices:1.3" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="urn:mtconnect.org:
    MTConnectDevices:1.3 /schemas/MTConnectDevices_1.3.xsd">
3 <Header creationTime="2017-03-03T15:28:13Z" sender="mtcagent" instanceId
    ="1485218744" version="1.3.0.9" assetBufferSize="1024" assetCount="0"
    bufferSize="131072"/>
4 <Devices>
5 <Device id="dev" iso841Class="6" name="VMC-3Axis" sampleInterval="10" uuid
    ="000">
6 <Description manufacturer="SystemInsights"/>
7 <DataItems>
8 <DataItem category="EVENT" id="avail" type="AVAILABILITY"/>
9 <DataItem category="EVENT" id="dev_asset_chg" type="ASSET_CHANGED"/>
10 <DataItem category="EVENT" id="dev_asset_rem" type="ASSET_REMOVED"/>
11 </DataItems>
12 <Components>
13 <Axes id="ax" name="Axes">
14 <Components>
15 <Rotary id="c1" name="C">
16 <DataItems>
17 <DataItem category="SAMPLE" id="c2" name="Sspeed" nativeUnits="REVOLUTION/
    MINUTE" subType="ACTUAL" type="SPINDLE_SPEED" units="REVOLUTION/MINUTE">
    <Source>spindle_speed</Source>
18 </DataItem>
19 <DataItem category="SAMPLE" id="c3" name="Sovr" nativeUnits="PERCENT" subType
    ="OVERRIDE" type="SPINDLE_SPEED" units="PERCENT">
```

Codeabschnitt 4.2: XML Informationsmodell einer Werkzeugmaschine

Stuctural Elements beschreiben die physischen, logische Komponenten und Subkomponenten von Devices. *Data Elements* beschreiben hingegen jene Daten, welche von einem Device gesammelt werden können.

In dem in Abbildung 4.12 dargestelltem Modell sieht man, dass ein oder mehr *Devices* (z.B. EMCO MT45) aus beliebig vielen Komponenten vom Typ *ComponentType* bestehen

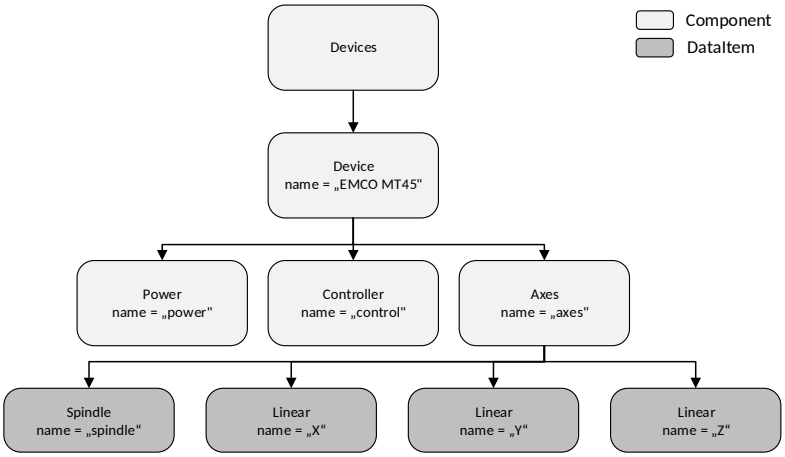

Abbildung 4.12: MTConnect - exemplarischer Aufbau des Informationsmodells für eine Werkzeugmaschine

können. Der Typ beinhaltet weitere Komponenten z.B. Controller, Linear Axis oder Rotatory Axis. Jede Komponente hat wiederum *DataItems*, welche vom Typ *Sample*, *Event* oder *Condition* sein können.
Sample-Werte (z.B. Xabs, Yabs) speichern numerische Werte als Zeichenkette (string). Event Tags haben vielfach Enumerationen zum Beschreiben der Zustände. Der *Controllermode* als Beispiel besitzt eine Enumeration mit den Werten: *MANUAL, MANUAL_DATA_ENTRY, AUTOMATIC*.
Zusätzlich benutzt MTConnect XML-Attribute, um das Informationsmodell noch weiter zu verfeinern. Beispiele für solche Attribute sind: *Name*, *Type*, *Subtype* und *Units*. In Abbildung 4.12 sind einige Attribute dargestellt.
Mit MTConnect Version 1.2 besteht die Möglichkeit zusätzlich herstellerspezifische Daten als XML Schema Definition darzustellen und mit MTConnect zu übertragen. Diese sogenannten *Assets* sammeln und melden komplette XML Dokumente, sobald sich diese ändern [80]. Vorerst spezifiziert MTConnect nur Werkzeuge (*CuttingTools*) als Assets [81] . Weitere Assets können jedoch jederzeit definiert werden.

In der aktuellen Spezifikation ist MTConnect nur als read-only Interface konzipiert, d.h., es können nur Daten gelesen werden. Normalerweise sendet ein Client eine HTTP-Get Anfrage an den Agent, welcher mit einer XML-Reply antwortet. Mit der Definition von MTConnect-Interfaces [79] ist es nun möglich, dem Agent Anfragen (Requests) zu senden. Die Spezifikation beschreibt das *BarFeederInterface*, *MaterialHandlerInterface*, *DoorInterface* und *ChuckInterface*. Diese können ähnlich wie die Assets jederzeit erweitert werden. Ein Beispiel für die Nutzung von Interfaces für den Datenaustausch bei der Werkzeugverwaltung wurden von Trautner et al. gezeigt [140].

4.6 Automation Markup Language (AutomationML)

Automation Markup Language (AutomationML) ist ein neutrales und XML-schemabasiertes Datenformat konzipiert für die herstellerunabhängige Speicherung von anlagentechnischen Informationen. Das Ziel von AutomationML ist die heterogene Werkzeuglandschaft der Engineering-Tools verschiedener Disziplinen, z.B. Maschinenbau, Elektrokonstruktion, HMI-Entwicklung, SPS, und Programmierung der Robotersteuerungen in ihrer Gesamtheit zu vernetzen [5], [29].

AutomationML ist in der Lage, logische Daten aus verschiedenen Werkzeugen und Disziplinen abzudecken und unterstützt verschiedene Phasen des iterativen Anlagenbaus mit unterschiedlichen Detaillierungsgraden. So können verschiedene Arten von Logik Informationen, die zu einer Industrieanlage oder zu einzelnen Komponenten gehören, gespeichert werden. Diese Informationsvielfalt lässt sich in zwei Hauptkonzepte unterteilen: *Sequencing* und *Behaviour*.

AutomationML bietet eine reichhaltige Auswahl an typischen Logikbeschreibungsmodellen für wichtige Phasen des Engineering-Prozesses [69], [5]:

- Gantt Diagramme für die ersten Planungsphasen
- PERT-Diagramme, die in ähnlicher Weise wie Gantt Diagramme bei komplexen Timing-Bedingungen verwendet werden
- Impulsdiagramme zur detaillierten Beschreibung von Sequenzen und zur Einführung realer Signale
- Zustandsdiagramme zur detaillierten Beschreibung des internen Verhaltens
- Sequenzfunktionsdiagramme (SFCs) für ausführbare SPS-Programme mit Abbildung auf reale Steuerungshardware

Das Datenformat AutomationML wurde von AutomationML e.V.[9] als Lösung für den Datenaustausch, mit der Fähigkeit, alle Informationen im Rahmen des Engineerings von Produktionssystemen abzudecken, entwickelt [69]. Es ist eine offenes, herstellerneutrales, XML-basiertes und freies Datenaustauschformat, das einen domänen- und firmenübergreifenden Engineering-Transfer ermöglicht.

AutomationML speichert technische Informationen nach dem objektorientierten Paradigma und erlaubt die Modellierung von physikalischen und logischen Anlagenkomponenten als Datenobjekte, die verschiedene Aspekte kapseln. Objekte können eine Hierarchie bilden, d.h., ein Objekt kann aus Unterobjekten bestehen und selbst ein Teil einer größeren Zusammensetzung oder Aggregation sein. Zusätzlich kann jedes Objekt Informationen über die im Objekt enthalten Eigenschaften von Geometrie, Kinematik und Logik

[9] www.automationml.org

(Sequenzierung, Verhalten, Steuerung und Informationen) sowie weitere Eigenschaften beinhalten.
AutomationML folgt einem modularen Aufbau durch Integration und Erweiterung bzw. Anpassung verschiedener bereits bestehender XML-basierter Datenformate unter einem Dach, das sogenannte Top-Level-Format.

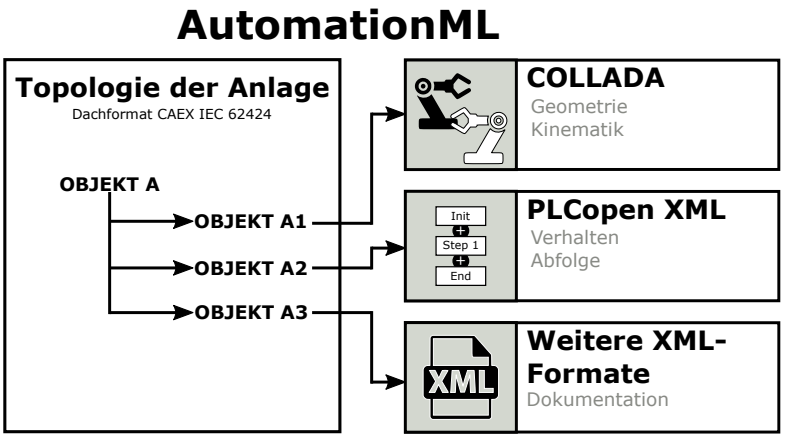

Abbildung 4.13: Datenstruktur AutomationML [12]

AutomationML ist in folgende Bereiche gegliedert (siehe Abbildung 4.13):

- Beschreibung der Komponententopologie und Netzwerkinformationen inklusive Objekteigenschaften ausgedrückt als Hierarchie von AutomationML-Objekten und beschrieben durch CAEX nach IEC 6242424 [45].
- Beschreibung der Geometrie und Kinematik der verschiedenen AutomationML-Objekte, dargestellt durch COLLADA [9].
- Beschreibung der steuerungsbezogenen Logikdaten der verschiedenen AutomationML-Objekte, dargestellt mittels PLCopen XML 2.0 und 2.0.1 [116].
- Die Beschreibung von Beziehungen zwischen AutomationML-Objekten und Verweisen auf Informationen, die sich in Dokumenten außerhalb des Top-Level-Formats befinden, erfolgt mittels CAEX.

4.7 Bewertung Modellierungsmöglichkeiten CPPS

Tabelle 4.1 zeigt eine Gegenüberstellung der in Kapitel 4 vorgestellten Modellierungssprachen. Die Bewertung erfolgt anhand der Fähigkeiten, die einzelnen Modelle für MDA bzw. die Modellierung von Information Interfaces für CPPS zu realisieren.

UML ist vor allem bei der Modellierung von generellen, abstrakten Informationen, wie sie bei der Erstellungen von Systemanforderungen verwendet werden, recht stark.

AutomationML kann hier ebenfalls einige Features aufweisen OPC UA und MTConnect sind für die Beschreibung von Anforderungen nicht geeignet.

Eigenschaft	UML	OPC UA	MTConnect	Automation ML
Anforderungen	●	○	○	◑
Plattformunabhängig	●	◑	○	○
Statische Strukturen	●	●	●	●
Zustandsautomaten	●	●	◑	●
Plattformspezifisch	○	●	●	●
Engineering	●	○	○	●
Information Interfaces	○	●	●	○

Tabelle 4.1: Gegenüberstellung der Modellierungssprachen UML, OPC UA, MTConnect und AutomationML

Bei der Erstellung von plattformunabhängigen Informationen eröffnet UML die meisten Möglichkeiten, da es in der aktuellen Spezifikation Version 2.5 [90] 14 Diagramme anbietet. Für die Modellierung von statischen Strukturen und Zustandsautomaten können alle Modelle verwendet werden. Nur MTConnect bietet keine Möglichkeit Zustandsautomaten abzubilden.

Sobald plattformspezifische Modelle benötigt werden, sind die generellen Modellierungskonzepte von UML zu wage, um spezifische Details in Modellen abbilden zu können. Bei der Erstellung von plattformspezifischen Modellen sind OPC UA, MTConnect und AutomationML besonders gut geeignet, da sie für die Modellierung konkreter spezifischer Probleme konzipiert sind.
Zusammenfassend kann gesagt werden, dass für die Modellierung von Information Interfaces von CPPS die UML und OPC UA alle benötigten Modellierungsaspekte abdecken. Für die Modellierung von anderen Systemsichten wie z.B die Darstellung von Engineering Daten ist eine Kombination aus AutomationML und OPC UA die bessere Wahl.

KAPITEL 5

Informationsmodellierung von Cyber-Physical Production Systems mit OPC UA

Wie in den vorherigen Kapiteln erklärt, sind Cyber-Physical Systems (CPS) vernetzte Systeme mit teils hohem Komplexitätsgrad (vgl. Kapitel 2.2). Um diese Vernetzung zu realisieren, gibt es unterschiedliche Kommunikationslösungen (Kapitel 3). Aufgrund der zahlreichen Fähigkeiten ist OPC Unified Architecture (OPC UA) eine der zukunftsträchtigsten.
Wie in Kapitel 3.1 erwähnt, basiert die Kommunikation via OPC UA auf einem Server-Client Prinzip. Der Server bietet seine Informationen bzw. Funktionen einem oder mehreren Clients an. Die Beschreibung der Informationen und Funktionen erfolgt mit einem Informationsmodell. In einem zweiten Schritt wird dieses Informationsmodell von einer Software, einem Server, instanziiert und somit die Funktionen des CPS anderen Systemen angeboten.

Für die Definition und Modellierung der Informationsmodelle wird Fachkompetenz und Wissen von unterschiedlichsten Experten aus unterschiedlichen Domänen benötigt. Gerade dieses Wissen stellt für viele Anwender eine Hürde dar, welche nur schwer erklommen werden kann. Hinzu kommen unzählige Standards, welche bei der Modellierung berücksichtigt werden müssen. Trotz einiger Referenzmodelle, wie z.B. dem Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0), ist dies eine große Herausforderung.

Um diesen Aufwand für die Implementierung zu reduzieren, werden oft Software Development Kit (SDK)s eingesetzt. Auch finden sich unzählige Anbieter auf dem Markt, welche wiederum spezielle Tools für die Unterstützung anbieten. Der Funktionsumfang einer solchen Software reicht von der graphischen Informationsmodellierung bis hin zur

Generierung von lauffähigem Code. Diese unterschiedlichen Eigenschaften erlauben es oft nicht, Projekte von einem Tool in ein anderes zu exportieren. Speziell der Export der Informationsmodelle in ein genormtes XML Format führt oftmals zu keiner Lösung, da unterschiedliche XML-Schemata verwendet werden.

Die Verwendung von Modellen gilt in einigen Branchen als Lösung für eine Komplexitätsreduktion. Modellgetriebene Ansätze, wie es Model Driven Architecture (MDA) ist, eignen sich besonders, da die Modellierung von der zu implementierenden Technologie separiert wird [85]. MDA ist nur ein Framework für die modellgetriebene Softwareentwicklung und beschreibt drei Phasen (CIM, PIM, PSM) vom Design zur Implementierung [60, 17]. Daher müssen die einzelnen Phasen für jeden einzelnen Anwendungsfall detailliert beschrieben und definiert werden [37]. Natürlich gibt es schon einige unterschiedliche Ansätze in unterschiedlichen Domänen, welche Diagrammtypen sinnvoll sind. Kriouile et al. [62] beschreiben eine Vielzahl von möglichen Diagrammtypen, welche für die Modellierung der Platform Independent Model (PIM) und Platform Specific Model (PSM) verwendet werden können.

Da es keine definierten Artefakte für die Modellierung von OPC UA Informationsmodellen für CPS gibt, müssen diese erst festgelegt werden. Dazu kommen noch einige Einschränkungen, welche die unterschiedlichen Technologien (z.B. OPC UA) mit sich bringen. Besonders die Modellierung und Implementierung von Informationsmodellen stellen für viele Anwender eine Herausforderung dar.

5.1 Modellierung von OPC UA Informationsmodellen

Für die Modellierung von OPC UA Informationsmodellen existieren bereits einige Lösungsbausteine, die im Nachfolgenden erklärt werden. Companion Standards sind wohl die bekannteste Möglichkeit OPC UA Informationsmodelle zu gestalten. Diese sind wie in Kapitel 4.4.3 beschrieben, Richtlinien für die Gestaltung von OPC UA Informationsmodellen. Jedoch gibt es nur für bestimmte Domänen standardisierte Modelle, z.B. SPS oder Werkzeugmaschinen , jedoch für den Großteil der Geräte in der Produktion fehlen solche Modelle. Im Zusammenhang mit CPS ist wohl die Spezifikation des Mappings von Automation Markup Language (AutomationML) zu OPC UA besonders interessant. Diese Spezifikation beschreibt die Transformation von AML Modellen in OPC UA Informationsmodelle [95]. Eine zweite Spezifikation ist vor allem für die Fertigungsdomäne interessant. Die MTConnect Comapanion Specification beschreibt, wie die MTConnect Datenstruktur in OPC UA abgebildet werden kann.

Zusätzlich zu diesen Standards existieren schon einige Bausteine im Bereich der Transformationen von UML Modellen und OPC UA Informationsmodellen. Rohjans et al. [118] haben einen Mechanismus für die automatische Transformation von UML Klassendiagrammen in OPC UA Adressraummodelle für Power Systems vorgestellt. Das von ihnen entwickelte Tool *CIMBaT* [119] generiert aus den definierten UML Diagrammen lauffähi-

gen Code. Diese Lösung betrachtet jedoch nur die Übersetzung einer domänenspezifischen Lösung und außerdem nur das statische Verhalten eines Systems.
Des weiteren präsentieren Lee et al. [64] ein Mapping und eine Transformation von UML Diagrammen in OPC UA Modelle. Dabei beschränken sie sich auf die statische Struktur d.h., UML Klassendiagramme. Für die Transformation von OPC UA auf UML setzen sie auf Query View Transformation (QVT). Dieser Ansatz ist generisch und erlaubt prinzipiell auch eine inverse Transformation von UML zu OPC UA. Dafür wurden jedoch noch keine Regeln definiert.

5.2 Systematic OPC UA Information Model Design

Im Zuge des FFG Projektes *OPC4Factory*[1] wurde ein systematisches Vorgehen für das Design von OPC UA Informationsmodellen definiert. Speziell wurde dieser Ansatz für die OPC UA Informationsmodellierung und Erstellung von OPC UA Servern im Bereich der Fertigungstechnik entwickelt und im Zuge der 49$^{\text{th}}$ Conference on Manufacturing Systems als *Systematic Approach for OPC UA Information Model Design* vorgestellt [113].

In der nachfolgenden Abbildung (5.1) wird ein auf MDA basierender Ansatz beschrieben, welcher für die Informationsmodellierung von statischen und dynamischen Verhalten von CPS verwendet werden kann. Zusätzlich erlaubt der modellgetriebene Gedanke eine einfache Erstellung einer Software über den gesamten Lebenszyklus eines CPS.

Die verwendete Methodik unterstützt den kompletten Prozess der OPC UA Informationsmodellierung, von der Systemanalyse zum Design, bis hin zur Codeerzeugung und deren Instanziierung in einem lauffähigen OPC UA. Im Sinne der MDA und CPS bzw. Cyber-Physisches Produktions System (CPPS) wurden folgende Aspekte berücksichtigt [17], [55], [74], [148]:

- Beschreibung der Systemanforderungen in einer abstrakten Art, ohne dabei auf Implementierungsaspekte einzugehen (user requirements, etc.)
- Beschreibung des Systemverhaltens, der gespeicherten Daten, etc. ohne auf technische oder technologische Details einzugehen.
- Gemeinsame Verständnisbasis für die Zusammenarbeit von Experten aus unterschiedlichen Domänen
- Ein durchgängiger Modellierungsansatz für eine Service-orientierte Architektur für CPS vom Design bis zum lauffähigen Code

OPC UA Information Model Design versucht die oben angeführten Punkte umzusetzen und dadurch eine erhöhte (i) Übertragbarkeit auf andere Systeme (ii) Interoperabilität und (iii) Wiederverwendbarkeit durch architekturbedingte Trennung der Aufgaben zu

[1] https://www.ift.at/forschungsbereiche/forschungsprojekte/opc4factory/

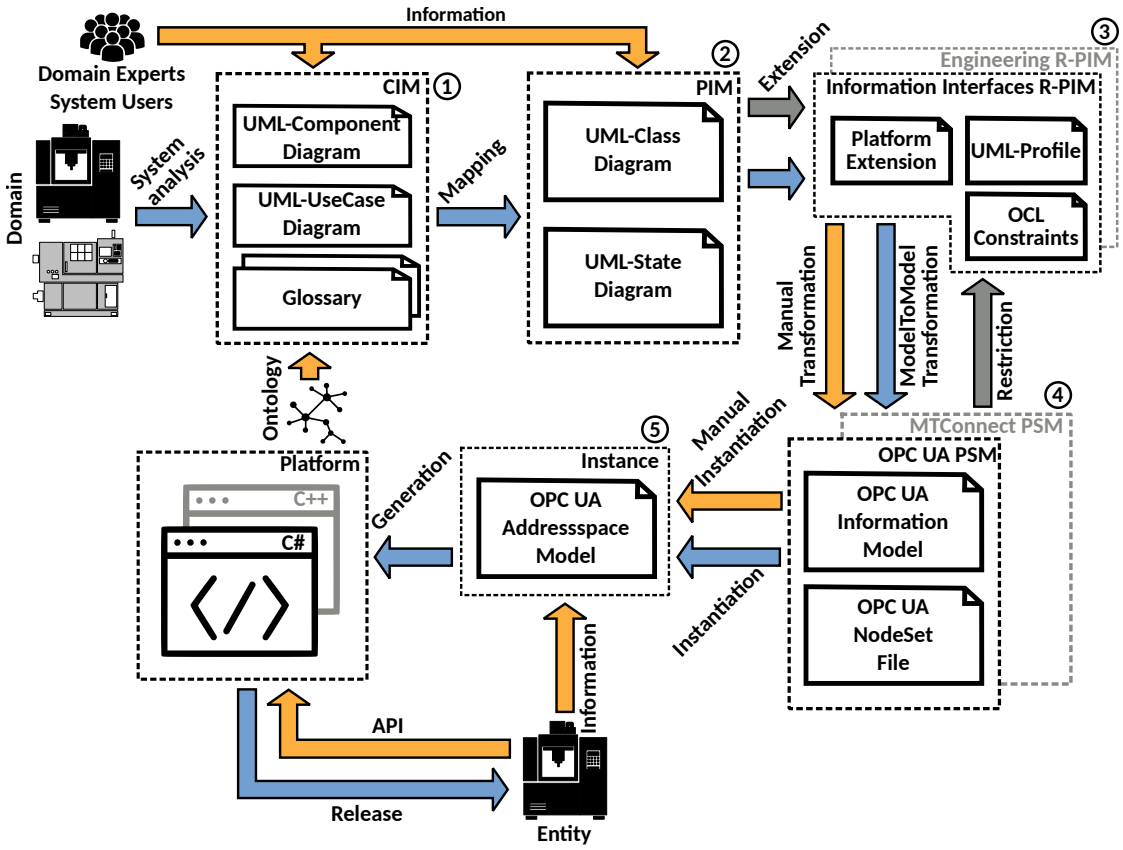

Abbildung 5.1: Vorgehen für die Informationsmodellierung von CPS mit OPC UA

erreichen. Deshalb wurden die drei Abstraktionslevels der MDA, das Computation Independent Model (CIM), das Platform Independent Model (PIM) sowie das Platform Specific Model (PSM) spezifiziert und durchlaufen. Ergänzt werden diese Phasen mit einer Code-Instanziierung sowie -Deployment Phase im Sinne der modellgetriebenen Softwareentwicklung [17].

Die strichlierten Rechtecke stellen die einzelnen Phasen des Erstellungsprozesses (CIM, PIM, R-PIM, PSM, Instanz und Plattform) von OPC UA Informationsmodellen dar. Die Objekte innerhalb der Phasen repräsentieren die einzelnen Modelle bzw. Artefakte (z.B. UML Klassendiagramm).

Die Pfeile stellen den Ablauf bzw. die Informationsflüsse dar. Die blauen Pfeile in Abbildung 5.1 repräsentieren die Arbeitsschritte, um OPC UA Informationsmodelle zu erstellen und anschließend in lauffähige OPC UA Server zu integrieren.
Die Interaktion von Benutzern und Experten in dem System wird als gelbe Pfeile dargestellt. Diese Interaktion können einerseits das Bereitstellen von Wissen oder Know-

How, anderseits das Ausführen gewisser Tätigkeiten sein. Aktuell ist or allem bei der Modellierung des Adressraummodells, sowie bei der Integration des Application Programming Interface (API) in den lauffähigen Code Expertenwissen notwendig. Zukünftig soll ein Eingriff von Menschen nur bei der Erstellung des Computation Independent Model (CIM) erfolgen, alle nachfolgenden Schritte könnten automatisiert ausgeführt werden.

Die grauen Pfeile in der Grafik zeigen die Restriktionen / Constraints. Diese unterscheiden den OPC UA Information Model Design Ansatz von einem klassischen MDA Ansatz. Damit kann frühzeitig in die Modellierung eingegriffen werden, da gewisse Modellkonzepte, welche nicht im jeweiligen PSM vorhanden sind, im PIM nicht verwendet werden. Da die Einschränkungen jedoch in Form von Modellen erfolgt gibt es weiterhin ein PIM, welches erst durch diese Restriktionen und Erweiterungen zum Restricted Platform Independent Model (R-PIM) wird. Das R-PIM wird im Detail in Kapitel 5.5 vorgestellt.

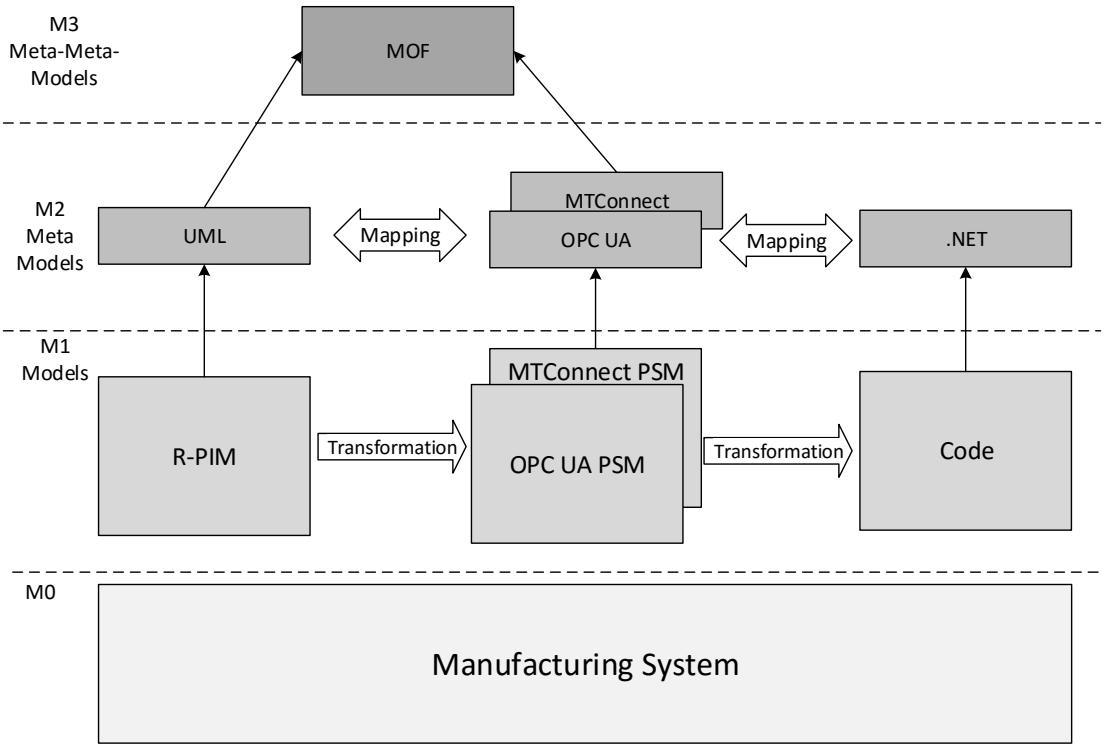

Abbildung 5.2: Transformationsprozess von PIM zu PSM

Hinter dem Ablauf, dargestellt in Abbildung 5.1, steckt das MDA Framework. Dass die einzelnen Modellelemente gemappt, und Modelle in andere Modelle transformiert werden können beruht auf dem *ModellingStack* von MDA. In Konformität zu [20] und [3] zeigt die Abbildung 5.2 die einzelnen Modellierungsebenen von MDA sowie den Ablauf, um ein plattformunabhängiges bzw. mit Restriktionen versehenes Modell in plattformspezifische

Modelle sowie letztendlich in ausführbaren Code zu transformieren.

Die Ebene M0 repräsentiert ein Fertigungssystem (Manufacturing System) als reales Objekt (Entität). Dieses System wird durch ein entsprechendes PIM, welches dem Unified Modelling Language (UML) Metamodell in Ebene M2 entspricht, modelliert. Somit wird ein plattformunabhängiges Modellieren auf der Ebene M1 ermöglicht. Das PIM als auch das PSM auf der Ebene M1 beschreiben das Fertigungssystem, selbst wenn diese in unterschiedlichen Sprachen bzw. Modellen beschrieben werden.

Die Transformation von PIM in ein PSM wird typischer Weise durch Transformation von einzelnen PIM-Elementen in PSM-Elemente durchgeführt. Ein einzelnes Element aus PIM kann durchaus in mehrere PSM Elmente übersetzt werden. Das PSM für OPC UA ist konform mit dem OPC UA Metamodell [100], andere PSMs müssen den entsprechenden Metamodellen konform modelliert werden (MTConnect Metamodell [77]).
Auf Code Ebene sieht das ganze ähnlich aus. Jede Applikation kann auf unterschiedlichen Systemen in unterschiedliche Programmiersprachen implementiert werden. So kann z.B. ein einzelnes PSM in unterschiedlichen Sprachen (z.B C++ oder C#) umgesetzt werden. Beim OPC UA Information Model Design werden beide Transformationsansätze Model-To-Text und Model-To-Model verwendet. Die M2M Transformation wird verwendet, um das OPC UA PSM zu erstellen. Eine detaillierte Beschreibung der Transformation ist im Kapitel 6 zu finden.
M2T wird herangezogen, um das zweite Artefakt des PSM, das NodeSet File zu bilden. Hier erfolgt eine Transformation der OPC UA Modelle in das XML Format. Des weiteren wird es dazu verwendet, um aus dem Adressraummodell lauffähigen Code zu erzeugen. Eine nähere Beschreibung ist in Kapitel 5.7 aufgeführt.

Damit ein Mapping auf der Modellebene M2 funktioniert, müssen alle Metamodelle einem gemeinsamen Meta-Metamodell entsprechen. Das MDA Framework sieht dabei das Meta-Object Facility (MOF) Modell vor [89]. Es garantiert die Interoperabilität der Metamodelle auf der M2 Ebene.

In den nachfolgenden Kapiteln werden die einzelnen Phasen/Abstraktionsstufen des OPC UA Information Model Designs näher beschrieben. Dabei stehen die beiden Abstraktionslevels PIM und PSM im Vordergrund. Es werden die benötigten Modelle, um das System zu beschreiben, sowie die Transformation zwischen diesen beschrieben. Dabei sollen die Modelle die zwei wichtigsten Eigenschaften eines Systems behandeln, den statischen (strukturellen) und den dynamischen Teil [17, 16]. Da das CIM durchaus sehr spezifisch auf die jeweilige Domäne ausfällt, wird kein Schwerpunkt in der Arbeit auf diese Phase gelegt. Selbiges gilt auch für die Umsetzung auf eine spezifische Plattform.

5.3 Computation Independent Model (CIM)

Das Computation Independent Model (CIM) wird vielfach auch als Business oder Domänenmodell verstanden, weil ähnliches Vokabular wie es Subject Matter Experts (SMEs) tun würden, verwendet wird. Ziel ist es, das System und dessen Funktionsweise zu beschreiben, ohne dabei auf technologiespezifische Aspekte einzugehen. Diese spezielle Sicht soll die Anforderungen an das System zu Tage bringen [2, 141]. Somit ist die erste Phase des OPC UA Information Model Designs eine detaillierte Systemanalyse bzw. Anforderungsanalyse. Das Ziel dieser Systemanalyse bzw. die Hauptaufgabe dieser sind laut [28], [14], [125]:

1. Die Abgrenzung des Systems
   In diesem Schritt sollen die Systemgrenzen festgelegt und Schnittstellen des Systems zur Umwelt definiert werden. Diesen Schritt könnte man auch als Eingrenzung auf eine Domäne bzw. ein Anwendungsgebiet verstehen.

2. Die Funktionen des Systems bestimmen
   In dieser Phase soll ein detailliertes Verständnis über die Funktionen des Systems geschaffen werden.

3. Die Modellierung eines Begriffsmodells
   Hier wird eine detaillierte Systembeschreibung erstellt. Dazu bedient man sich Modellen bzw. der Methoden der Systemmodellierung.

Entwirft man ein System, so empfiehlt es sich, die aus der Analyse gewonnenen Ergebnisse möglichst weiter zu verwenden. Dies ist dann gegeben, wenn man auf Notationsbrüche verzichtet (z.B. ein Modell der Analyse wird als Basis für das Modell des Designs verwendet und weiter konkretisiert und verfeinert) [123].
Um diese Anforderungen zu erfüllen, werden UML Use-Case- und Komponentendiagramme zur Definition des CIM verwendet. Zusätzlich zu den Diagrammen wird ein Glossar, mit den in der Domäne üblichen Begriffen definiert, um eine einheitliche Begriffsdefinition zu erhalten. Zusätzlich vereinfachen die in diesen Dokumenten definierten Begriffe die Modellierung des R-PIM.

Abgrenzung des Systems

UML Komponentendiagramme beschreiben die statische Struktur eines Systems und erlauben es Systemgrenzen und Sub-Systeme zu definieren. Bei der Systemzerlegung werden mehrere Abstraktionsebenen gebildet, die von sehr abstrakt bis zu sehr detailliert reichen können. Dabei ist ein Top-Down bzw. Bottom-Up Ansatz hilfreich. Mit Hilfe dieser Abstraktionsebenen kann die Systembeschreibung übersichtlich gehalten werden. Die ersten Ebenen sollen dazu dienen, das System abzugrenzen. Die detaillierten Betrachtungen sind eigentlich schon Bestandteil der Phase Systementwurf. Zusätzlich können die Schnittstellen und die auszutauschenden Daten zwischen den einzelnen Systemen bzw. Sub-Systemen definiert werden.

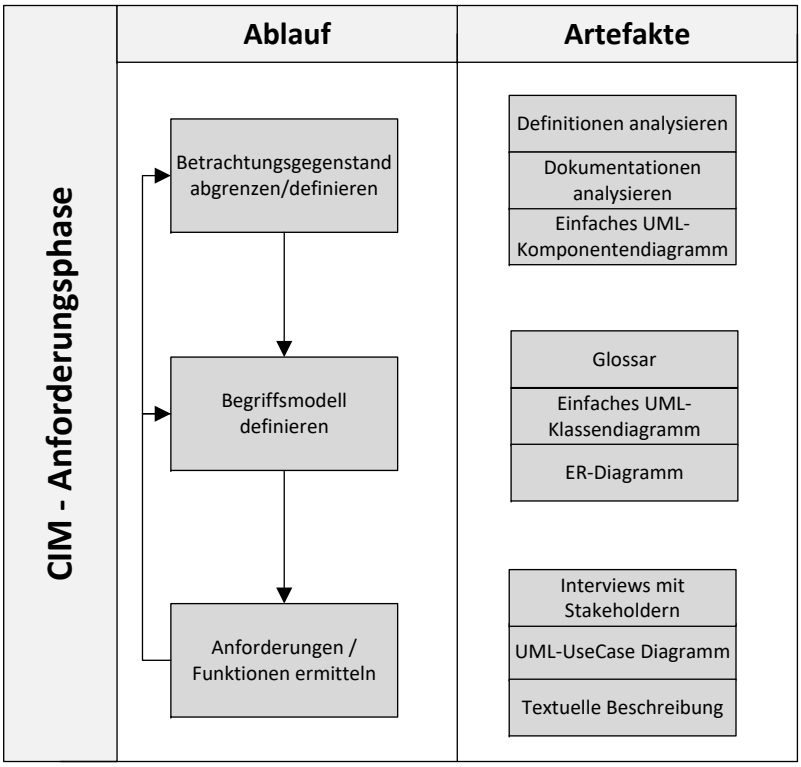

Abbildung 5.3: Anforderungsphase, Domain Description, CIM

Begriffsmodell

Ein großes Problem bei der Kommunikation zwischen zwei Individuen ist die Eindeutigkeit der Begriffe. Jeder Mensch besitzt ein individuelles Sprachverständis. Dies kann bei der Analyse eines Systems zu gravierenden Problemen führen. Deshalb ist es wichtig, ein Begriffsmodell zu definieren, damit alle Beteiligten ein gemeinsames Verständnis erwerben. Für ein Begriffsmodell gibt es unterschiedlichste Ausprägungen [125],[124], [115]. Im Folgenden werden die unterschiedlichen Varianten erklärt:

Glossar Die einfachste Form ist die Darstellung in Form eines Glossars. Der Glossar definiert alle speziellen Begriffe der Domäne, um Missverständnisse zu minimieren. Ebenfalls werden Synonyme, Akronyme und Abkürzungen erklärt. Begriffe, welche unterschiedliche Bedeutungen in der Domäne und der Umgangssprache haben, sollten ebenfalls erklärt werden [151].

Einfaches Klassendiagramm Das einfache Klassendiagramm liefert eine detaillierte Begriffsdefinition. Dabei werden Prozessworte im Zusammenhang eines bestimmten

Substantivs dargestellt. Zudem ist es eine grafische Darstellung des Betrachtungsgegenstandes [124].

Entity Relationship Diagramm Ein ER Diagramm kann ebenfalls für die Beschreibung des Betrachtungsgegenstandes verwendet werden [115]. Dies trifft dann zu, wenn eine Beschreibung des Zusammenhanges zwischen der Struktur und Beziehung der Informationen notwendig ist [67].

Die fachlichen Begriffe und deren Zusammenhänge, welche für die Definition des Begriffsmodells erforderlich sind, werden von den Anforderungen abgeleitet. Zusätzlich können weitere Informationen von Stakeholder durch Interviews oder aus Dokumentationen betrachtet werden.

Funktionale Beschreibung

Use-Case Diagramme werden verwendet, um die Funktionen des Systems in einer abstrakten Form zu beschreiben. Dadurch wird die Modellierung der dynamischen Struktur von CPS im R-PIM (Kapitel 5.5) vereinfacht, da die notwendigen und anzubietenden Methoden schon hier definiert wurden. Zum Identifizieren und Beschreiben der einzelnen Funktionalitäten gibt es unterschiedlichste Methoden wie die strukturierte Analyse (SA) [117] oder die objektorientierte Analyse [24]. Alle diese Methoden verfolgen das Ziel Benutzer des Systems (Akteure) sowie die Funktionen (Use-Cases) zu definieren. Besonders wichtig ist die Einhaltung der vorher definierten Systemgrenzen zur Vereinfachung der Analyse.

5.4 Platform Independant Model PIM

Neben dem R-PIM gibt es auch ein plattformunabhängiges Modell, welches das statische und dynamische Verhalten des Systems beschreibt.

Statische Beschreibung mit Klassendiagrammen

Mit Hilfe der Klassendiagramme werden einzelne Klassen, deren Attribute und Operationen sowie die Beziehungen zwischen den einzelnen Klassen beschrieben. Die Identifikation der Klassen, Attribute und Operationen erfolgt mit klassischen Methoden der Systemanalyse. Dabei sind die Object-Oriented Software Engineering (OOSE) [52] oder die Object Modeling Technique (OMT) [122] als bekannteste Methoden zu erwähnen. Abbildung 5.4 zeigt die unterschiedlichen Vorgehensweisen bei der daten- bzw. verhaltensorientierten Modellierung des Systems. Das Ergebnis dieser beiden ist ident, der Pfad jedoch unterschiedlich. Bei der Validierung in Kapitel 7 wurden die Klassendiagramme mit Hilfe der datenorientierten Methode erstellt.

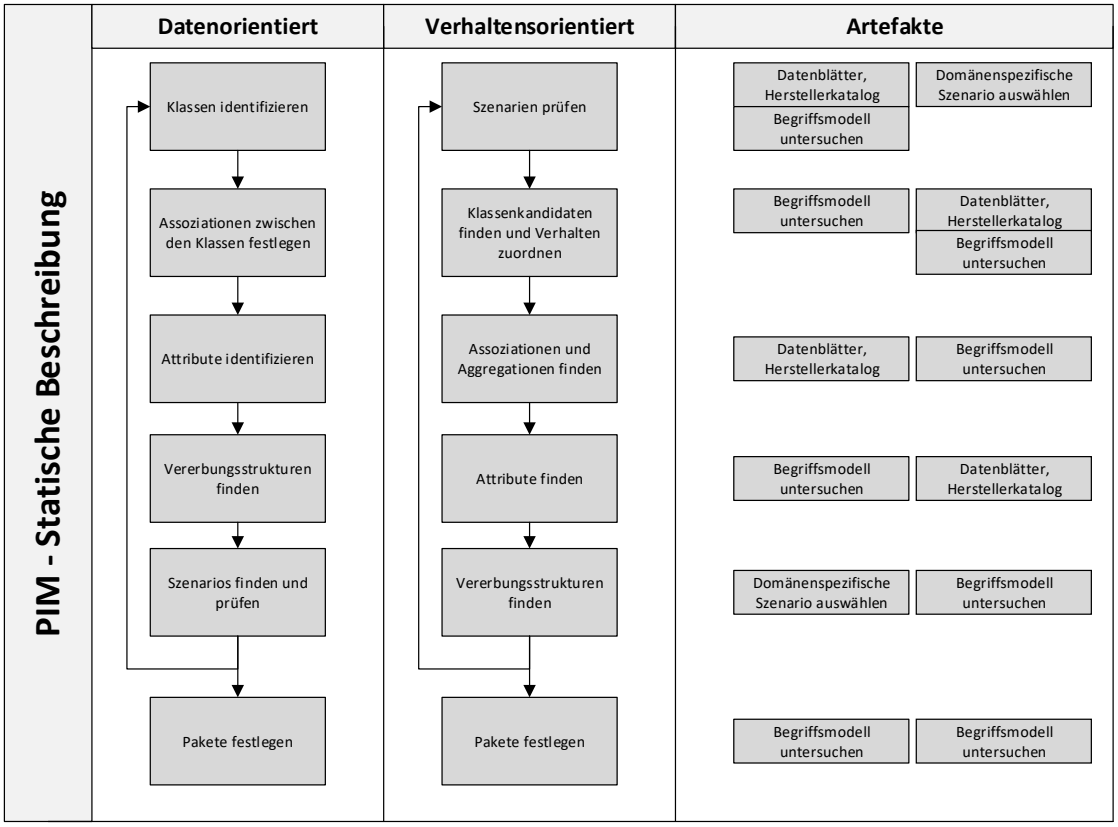

Abbildung 5.4: Vorgehen statische Modellierung für Daten orientierte und Verhaltens orientierte Systeme

Dynamische Beschreibung mit State Charts

Die meisten Fertigungssysteme arbeiten Event-basiert, d.h., sie warten auf das Auftreten von externen oder internen Events. Nach dem Auftreten eines solchen werden spezifische Aktionen, wie z.B. das Öffnen einer Türe durchgeführt. Mit Hilfe von rollenbasierten Szenarien im CIM (siehe Kapitel 5.3) können die Subsysteme identifiziert werden. Das dynamische Verhalten dieser Systeme kann mit UML-State Charts detailliert beschrieben werden. Dabei werden die einzelnen Zustände, sowie die Zusammenhänge mit Übergangsbedingungen zwischen diesen beschrieben [125].
UML State Machines zeigen die Zustände, die ein Objekt im Lauf seines Lebens einnehmen kann. Durch Ereignisse (Events) werden Zustandsübergänge (Transitions) ausgelöst. Sie werden bevorzugt verwendet, um asynchrone Vorgänge abzubilden. Abbildung 5.5 zeigt die Vorgehensweise, wie Zustandsdiagramme erstellt werden. Dabei werden am Anfang die Zustände des Systems definiert, im nächsten Schritt die Übergänge zwischen den einzelnen Zuständen. Hier werden auch die Namen für die Transitionen festgelegt. Im letzten Schritt werden die Bedingungen für einen erfolgreichen Übergang (Guards) zwischen

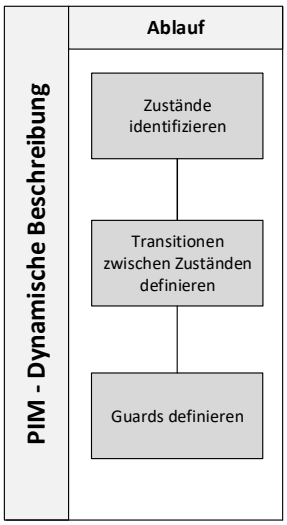

Abbildung 5.5: Vorgehen bei der Modellierung des dynamischen Systemverhaltens

zwei Zuständen definiert.

5.5 Restricted Platform Independent Model (R-PIM)

Die dritte Phase des OPC UA Information Model Design, die plattformunabhängige Modellierung, basiert auf der Verwendung von UML Klassen- und UML Zustandsdiagrammen um das statische und dynamische Verhalten von CPS zu beschreiben. Wie in Kapitel 5.2 beschrieben, verwendet dieser Ansatz ein eingeschränktes PIM. Die Einschränkungen beziehen sich hier auf die Plattformunabhängigkeit. In Abbildung 5.1 wir der Unterschied zwischen einem klassischen MDA Ansatz und dem OPC UA Information Model Design deutlich.

Beim klassischen MDA gibt es eine strikte Trennung zwischen PIM und PSM. Beim OPC UA Information Model Design ist dies nicht der Fall. Es gibt das R-PIM, welches einerseits das PIM um Informationen für das PSM erweitert und anderseits das PIM in seinen Möglichkeiten einschränkt (siehe graue Pfeile in der Abbildung). Das R-PIM wird spezifisch für die unterschiedlichen Systemsichten erstellt. Für Information Interfaces wird ein anderes R-PIM als für das Engineering definiert.

Bei der Modellierung von OPC UA Informationsmodellen hat sich herausgestellt, dass einige Konzepte, welche in UML verwendet werden können, in OPC UA verboten sind. Deshalb wurden semantische Restriktionen des PIM definiert. Diese wurden in der computerauswertbaren, plattformunabhängigen Sprache Object Constraint Language (OCL) definiert, da sie gemeinsam mit UML spezifiziert wurde.

Die Restriktionen gelten für eine Systemsicht. Beim OPC UA Information Model Design wurde die Kommunikationsfähigkeit von CPPS betrachtet. Dazu wurden einige der detailliertesten TCP-basierenden Kommunikationstechnologien wie z.B. MTConnect, Message Queue Telemetry Transport (MQTT) oder eben auch OPC UA untersucht, und ein Set von Restriktionen definiert. Ein Beispiel für eine Restriktion sind Mehrfachvererbungen. Diese können zwar in Konformität zur Definition in UML modelliert werden, jedoch gibt es diesen Mechanismus in kaum einer Implementierungssprache. Eine Auflistung von Einschränkungen von UML für die Modellierung von OPC UA Informationsmodelle werden in Kapitel 6.4 genauer beschrieben.

Neben den Restriktionen ist es auch sinnvoll, das PIM durch Informationen, welche bei der platformspezifischen Modellierung notwendig sind, zu ergänzen (Extension Pfeil in Abbildung 5.1). Das PIM mit zusätzlichen Informationen zu versehen, ist durchaus üblich. Cortellessa et al. [25] ergänzen ihr PIM mit Leistungsdaten. Grundsätzlich führen die Anreicherung des PIM zu einer einer höheren Reproduzierbarkeit bei der Transformation von PIM in ein PSM [147]. Die Erweiterung der UML Funktionalität wird mit Hilfe eines UML-Profils vorgenommen. Dies trifft nur auf die statische Beschreibung mit UML-Klassendiagrammen zu. Dort werden einzelne Klassen und Attribute durch zusätzliche Eigenschaften, welche für die Transformation in ein PSM notwendig sind, ergänzt. Eine Erweiterung der UML Zustandsdiagramme ist für die Informationsmodellierung nicht notwendig.
Ein Beispiel für die Erweiterung sind die Attribute eines jeden OPC UA Knotens. Einige können natürlich aus den Informationen der Klassen und deren Attributen entnommen werden, die restlichen Merkmale müssen in Form eines Profils ergänzt werden, um den Anforderungen von OPC UA gerecht zu werden. Eine detaillierte Beschreibung der Erweiterung von UML ist in Kapitel 6.2 zu finden.

5.6 Platform Specific Model (PSM) für OPC UA

Das PSM erweitert das allgemeine PIM um Aspekte, die rein plattformspezifisch sind, in diesem Fall OPC UA. Das PSM Modell beinhaltet alle wichtigen Information, welcher ein OPC UA Server benötigt, um seine Informationen einem Client anzubieten Das PSM besteht aus 2 Artefakten. Einerseits dem graphischen OPC UA Modell, welches mit Hilfe der Transformationsregeln erstellt wurde. Anderseits wird auch ein XML Dokument, das NodeSetFile erstellt. Dies erlaubt es, dass das OPC UA Informationsmodell maschinenlesbar weitergegeben und verarbeitet werden kann.

In dieser Phase werden OPC UA Modelle entsprechend der graphischen Notation verwendet. Bei der Modellierung werden folgende Konzepte aus OPC UA berücksichtigt, das Basisinformationsmodell (OPC UA Spezifikation Part 5 [102]) und die Erweiterung mit State Machines (Annex B OPC UA Spezifikation Part 5 [102]).
Ein Beispiel für ein Informationsmodell in der graphischen Notation ist in Kapitel 4.4.4 Abbildung 4.10 dargestellt.

Das zweite Artefakt des PSM ist die NodeSet Datei. Annex F im Part 6 der OPC UA Spezifikation [103] beschreibt einen standard Syntax, zur Definition von OPC UA Informationsmodellen. Dieses OPC UA Information Model XML Schema erlaubt ein computerisiertes Auslesen eines OPC UA Informationsmodells. In Codeabschnitt 5.1 ist ein Ausschnitt aus einem NodeSet File abgebildet.

```
1 UANodeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
      http://www.w3.org/2001/XMLSchema" xmlns="http://opcfoundation.org/UA
      /2011/03/UANodeSet.xsd" Version="1.02" LastModified="2013-03-06T05
      :36:44.0862658Z">
2 <Aliases>...</Aliases>
3 <UAObject NodeId="i=3062" BrowseName="Default Binary" SymbolicName="
      DefaultBinary">
4    <DisplayName>Default Binary</DisplayName>
5    <Description>The default binary encoding for a data type.</Description>
6    <References>
7       <Reference ReferenceType="HasTypeDefinition">i=58</Reference>
8    </References>
9 </UAObject>
10 <UAObject NodeId="i=3063" BrowseName="Default XML" SymbolicName="DefaultXml">
11    <DisplayName>Default XML</DisplayName>
12    <Description>The default XML encoding for a data type.</Description>
13    <References>
14       <Reference ReferenceType="HasTypeDefinition">i=58</Reference>
15    </References>
16 </UAObject>
17 ...
```

Codeabschnitt 5.1: OPC UA Node Set File

5.7 Instanziierung, Codegenerierung und Deployment

Die Instanziierung erfolgt manuell. Dabei wird aus dem Informationsmodell durch Instanziierung ein Adressraummodell erstellt. Dieses repräsentiert dann nur mehr eine Entität. Im Anschluss an die Instanziierung erfolgt die Codegenerierung. Bei der Codegenerierung wird aus dem Adressraummodell ein lauffähiger Code generiert. Dieser wird anschließend mit den realen Objekten, deren Zustände und Informationen verbunden.

Für beide Phasen wird auf bereits bestehende Software gesetzt, da hierfür schon einige Tools auf dem Markt verfügbar sind. Diese Tools besitzen unterschiedliche Funktionsumfänge, vom einfachen Modellieren des Adressraumes durch Hinzufügen von Knoten (Nodes) und Referenzen (References) bis hin zur Generierung von lauffähigen Applikationen (siehe Abbildung 5.6).

Abbildung 5.6 zeigt die Funktionen eines typischen Modellierungstools. Alle Tools erlauben das Einlesen von Informationsmodellen. Dies können entweder Companion Standards

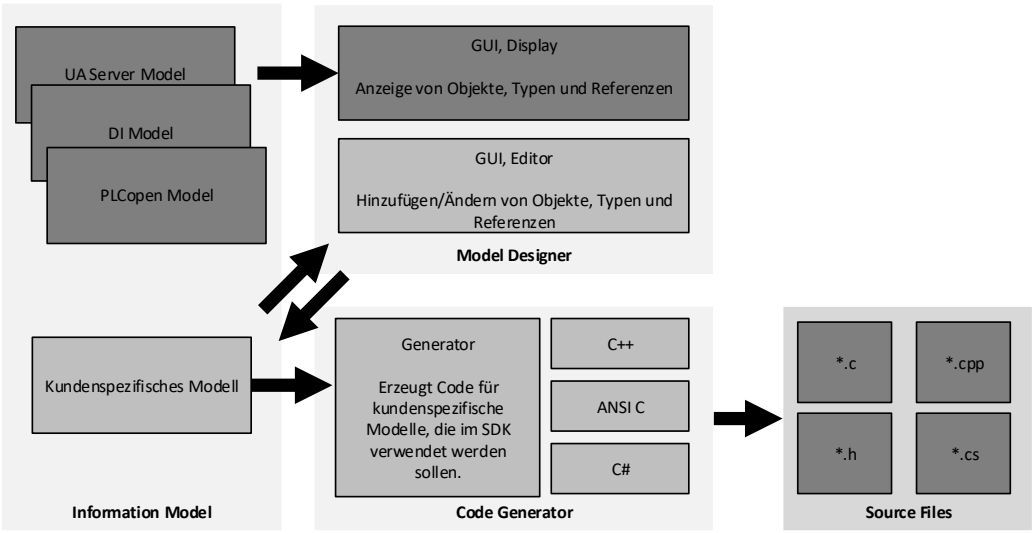

Abbildung 5.6: Funktionen von existierenden OPC UA Modellierungswerkzeugen[2]

(PLCopen) oder selbst erstellte Modelle sein (Information Model). Meist verfügen diese Tools über eine graphische Oberfläche, welche es erlaubt die Modelle zu editieren und anzuzeigen (ModelDesigner). Eine weitere Funktion ist die Code Generierung. Hier werden die vorher erstellten oder importierten Modelle in lauffähigen Code transformiert. Dabei kann hier die zu implementierende Plattform (meist C, C++, C# oder Java) ausgewählt werden. Ergebnis dieser Codegenerierung sind eine definierte Anzahl von Quelldateien, welche einen lauffähigen OPC UA Server repräsentieren.
Aktuell gibt es eine Vielzahl von Software Entwicklungspaketen (SDK), welche für die Realisierung eines OPC UA Servers verwendet werden können. Die bekanntesten sind jene von der OPC Foundation[3] und von Unified Automation[4]. Jedes dieser SDKs benötigt einen spezifischen Code, damit ein lauffähiger OPC UA Server erstellt werden kann. Alle diese entsprechen dem Model-Driven Software Development (MDSE) und können im OPC UA Information Model Design verwendet werden.

Abschließend ist wieder ein User gefragt. Dieser muss nun den OPC UA Server mit realen Daten der Maschine befüllen (gelber Information Pfeil in Abbildung 5.1). In Codeabschnitt 5.2 ist ein Ausschnitt eines automatisch generierten Codes zu sehen. Die Zeilen 1-10 werden von dem Tool automatisch generiert. Die Zeilen 12-24 müssen vom User manuell eingetragen werden und befüllen den OPC UA Server mit realen Daten.

[2] https://www.unified-automation.com/products/development-tools/uamodeler.html
[3] https://opcfoundation.org/products/view/net-based-opc-ua-server-sdktoolkit
[4] https://www.unified-automation.com/

```
1 public override void Startup()
2 {
3    try
4    {
5       Console.WriteLine("Starting IRBNodeManager.");
6       DefaultNamespaceIndex = AddNamespaceUri("http://yourorganisation.org/
            OPCUAIRB/");
7       Console.WriteLine("Loading the IRB Model.");
8       ImportUaNodeset(Assembly.GetEntryAssembly(), "opcuairb.xml");
9       CreateObjectSettings settings;
10      ObjectNode node;
11
12      //RobotController --> User Input
13      #region RobotController
14         //link objects
15         RobotControllerModel irbcontroller = new RobotControllerModel(
               controller);
16         LinkModelToNode(ObjectIds.RobotController.ToNodeId(Server.
               NamespaceUris), irbcontroller, null, null, 500);
17         irbcontroller.ControllerName = controller.Name;
18         irbcontroller.RobotWareVersion = controller.RobotWareVersion.
               ToString();
19         irbcontroller.SystemName = controller.SystemName;
20         irbcontroller.SystemID = controller.SystemId.ToString();
21         irbcontroller.SystemIPAddress = controller.IPAddress.ToString();
22         irbcontroller.SystemIPAddress = controller.MacAddress.ToString();
23         irbcontroller.State = controller.State.ToString();
24      #endregion
25   ...
```

Codeabschnitt 5.2: OPC UA generierter Code

5.8 Validierung der generierten Modelle

Für die Validierung der Transformation von Modellen gibt es eine Vielzahl von Möglichkeiten. Calegari und Szasz [22] haben eine Auflistung möglicher Verifikationen beschrieben. Um die Konsistenz des OPC UA Modells zu überprüfen, wird es auf folgenden Invarianten hin geprüft:

- Alle Instanzen entsprechen einem Typ
- Alle Typen sind Vererbungen von den OPC UA BaseTypes
- Alle Namen von Typen sind eindeutig und einzigartig
- Dem Namen Attribut ist ein Wert zugewiesen

Diese Invarianten können beliebig erweitert werden. Deren Definition erfolgt ebenfalls in der Object Constraint Language (OCL) [87].

Die Vorteile von OPC UA Information Model Design liegen in der Möglichkeit die formale Korrektheit der Modelle sicherzustellen. Die in der OPC UA Spezifikation definierten Regeln können mit OCL Bedingungen abgebildet und im Zuge der Transformation überprüft werden.

KAPITEL 6

UML2OPCUA

Die Transformation (Methode Abbildung 5.2) stellt neben den Modellen den wichtigsten Bestandteil der MDSE dar [17]. Um die Vision von MDSE wahr werden zu lassen, sollten die Transformationen automatisiert erfolgen [132]. Erst durch diese Automatisierung wird eine Komplexitätsreduktion bei der Informationsmodellierung realisiert. Teile der Transformation wurden im Zuge der 11th Conference on Intelligent Computation in Manufacturing Engineering vorgestellt [114].
Im nachfolgenden Abschnitt werden die dafür notwendigen Übersetzungsregeln, d.h., das Mapping von UML auf OPC UA definiert. Dieses besteht in der Definition von Zusammenhängen zwischen Elementen zweier Modelle, konkret der Relation zweier Metamodelle. Abbildung 6.1 zeigt die Zusammenhänge zwischen UML und OPC UA. Die Punkte in der Grafik stellen Elemente der jeweiligen Modelle dar.

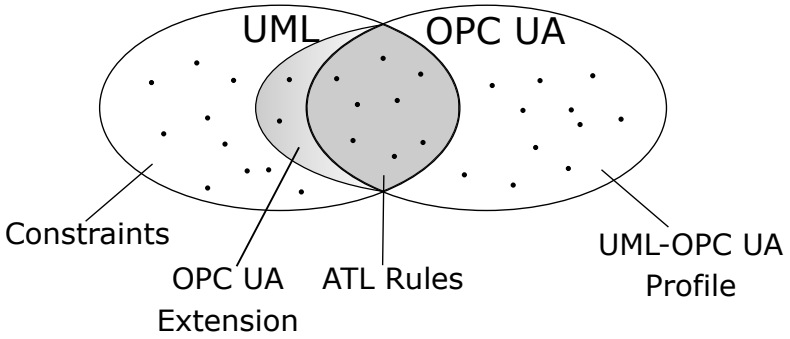

Abbildung 6.1: UML 2 OPC UA Konzept [114]

In der Schnittmenge befinden sich diejenigen Elemente, welche durch Transformationsregeln in die jeweils andere Sprache übersetzt werden können. Elemente, welche von dem Ursprungsmodell nicht in das Zielmodell gemappt werden können, müssen über

sogenannte Constraints/Restrictions abgefangen werden. Diese Restrictions schränken die Modellierungsmöglichkeiten des Ursprungsmodells ein. Außerdem können diese Elemente auch durch die Erweiterung von OPC UA abgebildet werden. Zusätzlich können Elemente, welche im Zielmodell notwendig sind, vom Ursprungsmodell aber nicht angeboten werden, mit Hilfe einer Erweiterung des Metamodells hinzugefügt werden. In UML spricht man bei dieser Form der Erweiterung des Metamodells von einem Profil. Diese Adaptionen werden in einer zusätzlichen Phase zwischen PIM und PSM vorgesehen, dem R-PIM.

6.1 Mapping zwischen UML und OPC UA

Das Mapping erfolgt immer auf der Metamodellebene, der Transformationsprozess findet dann auf der Modellebene statt (siehe Abbildung 6.2). Das hier beschriebene Mapping fokussiert sich auf Elemente von UML Klassen und State Diagrammen und deren Abbild in OPC UA. Das Besondere bei dieser Transformation ist, dass in OPC UA neben Typen auch Instanzen im Informationsmodell verwendet werden, d.h, dass einige UML Elemente in Typen, andere wiederum in Instanzen umgewandelt werden.

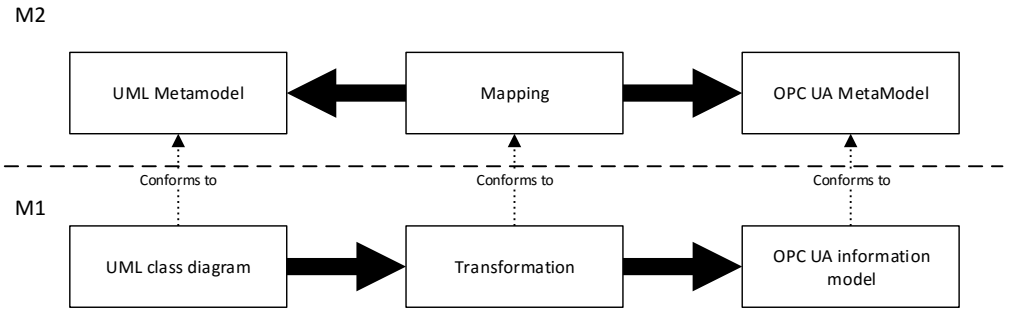

Abbildung 6.2: Mappingprozess zwischen UML und OPC UA [114]

Tabelle 6.1 gibt eine komplette Übersicht über alle Elemente, welche mit Mappingregeln abgebildet werden können. Zusätzlich ist noch angegeben, ob ein direktes Mapping erfolgt, oder ob die Attribute von anderen Elementen und Attributen abgeleitet wurden.

6.1.1 Klassendiagramm2OPCUA

NamedElement2Node

In OPC UA werden alle NodeClasses Elemente von einer Basisklasse abgeleitet, der Klasse *Base* (siehe Abbildung 4.9 in Kapitel 4.4.1). Diese Basisklasse besitzt die Attribute *NodeId*, *NodeClass*, *DisplayName*, *BrowseName*, *Description*, *WriteMask* und *UserWriteMask*, wobei einige davon optional sind.

In UML gibt es ebenfalls eine Klasse, von der alle anderen Elemente abgeleitet sind, die

UML Concept	OPC UA Concept	Comment
NamedElement	Node	mapped
	NodeID	<<NamedElement>>.name
	BrowseName	<<NamedElement>>.name
	DisplayName	<<NamedElement>>.name + <<BasicAttributes>>.Locale
	Description	<<BasicAttributes>>.Description
	UserWriteMask	<<BasicAttributes>>.AccessLevel
	WriteMask	<<BasicAttributes>>.AccessLevel
Class	UAObjectType	mapped
	NodeClass	derived (ObjectType)
	BrowseName	overriden (<<NamedElement>>.name+“Type“)
	DisplayName	overriden (<<NamedElement>>.name+“Type + <<BasicAttributes>>.Locale)
isAbstract	isAbstract	mapped
superClass	HasSubtype	inverse
ownedOperation	HasComponent	mapped
ownedAttribute	HasProperty HasComponent	mapped
Property	UAVariable	mapped (<<AdditionalAttributes>>.Behaviour)
	UAProperty	mapped (<<AdditionalAttributes>>.Behaviour)
	NodeClass	mapped (Variable)
	HasModellingRule	<<AdditionalAttributes>>.ModellingRule
Operation	UAMethod	mapped
	HasModellingRule	<<AdditionalAttributes>>.ModellingRule
	NodeClass	mapped (Method)
	ParentNodeID	derived (NodeId from the related Class)
DataType	UADataType	mapped
	NodeClass	mapped (DataType)
Enumeration	UADataType	mapped
PrimitiveTye	UADataType	mapped
Association	UAReference	
	NodeClass	mapped (Reference)
Generalization	HasSubtype	mapped
Association	NonHierarchic Reference	derived
Composition	HasComponent	mapped
Aggregation	Organize	mapped
MultiplicityElement	HasModellingRule	Derived
StateMachine	StatemachineType	Mapped
	BrowseName	overriden (<<NamedElement>>.name+“StateMachineType“)
	DisplayName	overriden (<<NamedElement>>.name+“StateMachineType + <<BasicAttributes>>.Locale)
State	State	mapped
Transition	UAReference	mapped
target	ToState Reference	mapped
source	FromState Refernce	mapped

Tabelle 6.1: Übersicht Mapping Elemente

Klasse *Element*. Die Klasse *NamedElement* ist eine Subklasse davon und besitzt nur ein Attribut, nämlich *Name*. Weitere Attribute werden durch das Profil, welches in Kapitel 6.2 beschrieben wird, ergänzt.

Somit kann die Klasse *NamedElement* und deren Attribute auf die Klasse *Base* gemappt

werden. Abbildung 6.3 zeigt die beiden Metamodelle von UML und OPC UA, welche diesen Ausschnitt darstellen.

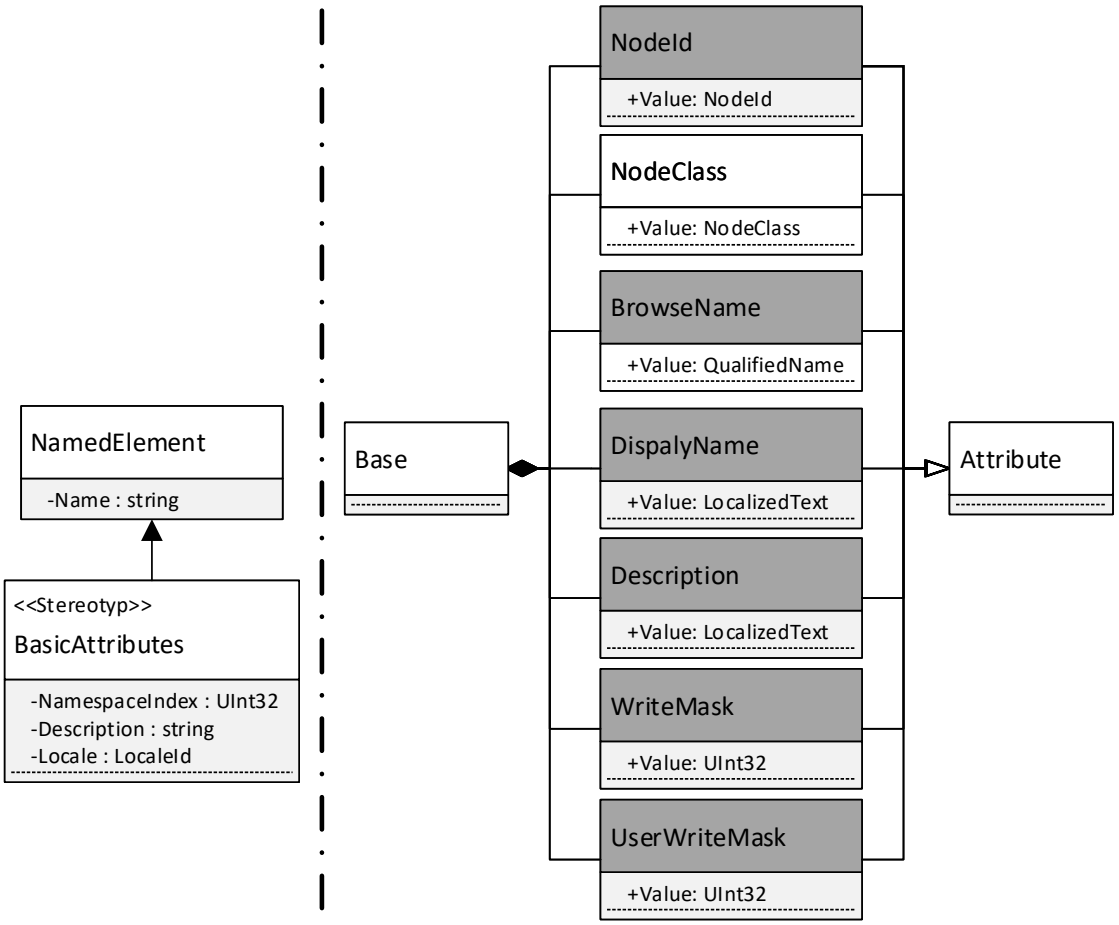

Abbildung 6.3: Mapping von UML Element zu OPC UA Node

Folgende Mappingregeln kommen dabei zum Tragen.

- Die *NodeId* in OPC UA ist ein komplexer Datentyp. Er besteht aus einem *NameSpaceIndex*, einem *IdentifierType* und einem *Identifier* (siehe Abbildung 6.1). Der *NameSpaceIndex* wird anstelle der NameSpace URI verwendet. Das NameSpaceArray beinhaltet alle NameSpace URIs und der NameSpaceIndex verweist auf eine Position in dem Array. Der NameSpaceIndex wird im Profil angegeben *«BasicAttributes».NameSpaceIndex*. In den nachfolgenden Beispielen wird diese als 1 angenommen.

```
1    "1",String, "path1/tag1"
```

Codeabschnitt 6.1: Beispiel Node

Für die Transformation wird der *«Element».Name* als Identifier verwendet. Als *IdentifierType* wird immer *String* verwendet und als *NameSpaceIndex* wird *1* angenommen.

- Der *BrowseName* ist vom Typ QualifiedName, einem komplexen Datentypen bestehend aus dem *NameSpaceIndex* und einem *Namen*. Der *NameSpaceIndex* wird wie bei der *NodeId* aus dem Profil übernommen. Der *Name* wird vom *«NamedElement».Name* übernommen.
- Der *DisplayName* ist vom Typ *LocalizedText*, welcher ebenfalls ein komplexer Datentyp bestehend aus einem *Text* vom Typ *string* und einer Ennumeration *Locale* vom Typ *LocaleId*. In der Transformation ist der *Text* gleich dem *«NamedElement».Name* und die *LocaleId* entsprcht der UML *«BasicAttributes».Locale*.
- Das Attribut *Description* entspricht dem *«BasicAttributes».Description* aus dem Profil.
- Die Attribute UserWriteMask und WriteMask werden entsprechend dem hinterlegten Wert des Attributs *«BasicAttributes».AccessLevel* beschrieben.
- Das NodeClass Attribut wird nicht in dieser Regel beschrieben, sondern erst in den nachfolgenden (Class2ObjectType, Attribute2Variable, Operation2Method).

Class2ObjectType

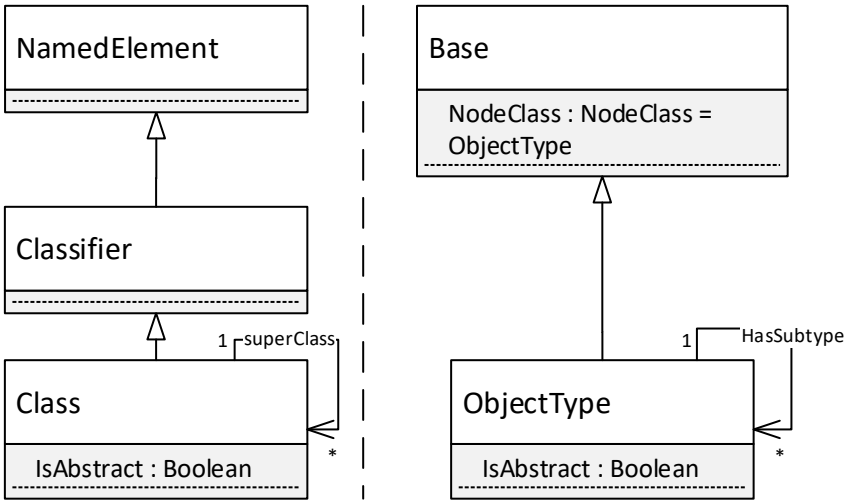

Abbildung 6.4: Mapping UML Klasse und OPC UA Object bzw. ObjectType

Eine UML Klasse (*Class*) entspricht einem OPC UA *ObjectType*. Abbildung 6.4 zeigt den Zusammenhang zwischen dem Metamodellen von UML und OPC UA. Eine Klasse (*Class*) ist ein Subtyp der Klasse *Classifier*, welche wiederum ein Subtyp von

NamedElement ist. In OPC UA ist der *ObjectType* ein Subtyp von *Base*.

Zusätzlich zu den Attributen, welche durch die Regel Element2Node gemappt werden, werden hier folgende Attribute beschrieben:

- *«Base».NodeClass* wird auf *ObjectType* gesetzt.
- Der Wert des Attributes *«ObjectType».IsAbstract* wird dem Wert *«Class».IsAbstract* gleichgesetzt.
- Der *DisplayName* und *BrowseName* wird gleich *«Class».Name* gesetzt

Attribute2Variable

UML *Attributes* entsprechen OPC UA *Variables*. Abbildung 6.5 zeigt den vereinfachten Zusammenhang zwischen den beiden Elementen in UML und OPC UA. Da es jedoch

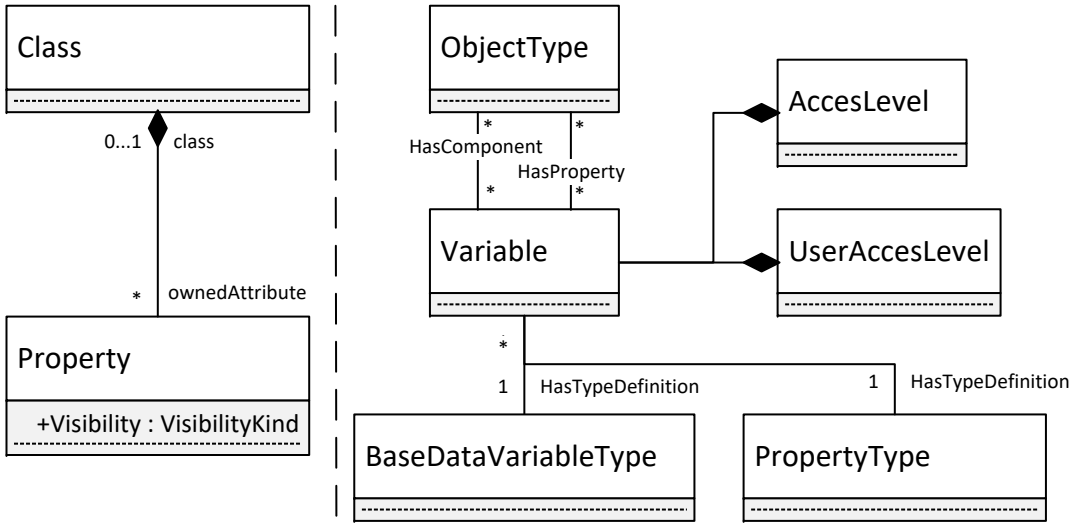

Abbildung 6.5: UML Attributes und OPC UA Variables

in OPC UA zwei Arten von Variablen gibt, muss für eine exakte und nachvollziehbare Übersetzung die Modellierung in OPC UA klar definiert werden. In OPC UA unterscheidet man *DataVariables* und *Properties*.

OPC UA Properties werden verwendet, um die einen Knoten zu charakterisieren. Ihre Werte ändern sich typischerweise nicht und werden üblicherweise in einer Konfiguration oder Datenbank gespeichert und in der Serverinitialisierung gelesen und beschrieben.
OPC UA DataVariables werden verwendet, um Daten von Objekten zu repräsentieren, welche sich häufig ändern. DataVariables können auch dazu verwendet werden, um komplexe Datenstrukturen abzubilden. Konkret können sie Subvariablen oder Properties wie z.B. EngineeringUnits besitzen. Die Transformation von UML Attributen muss mit dieser

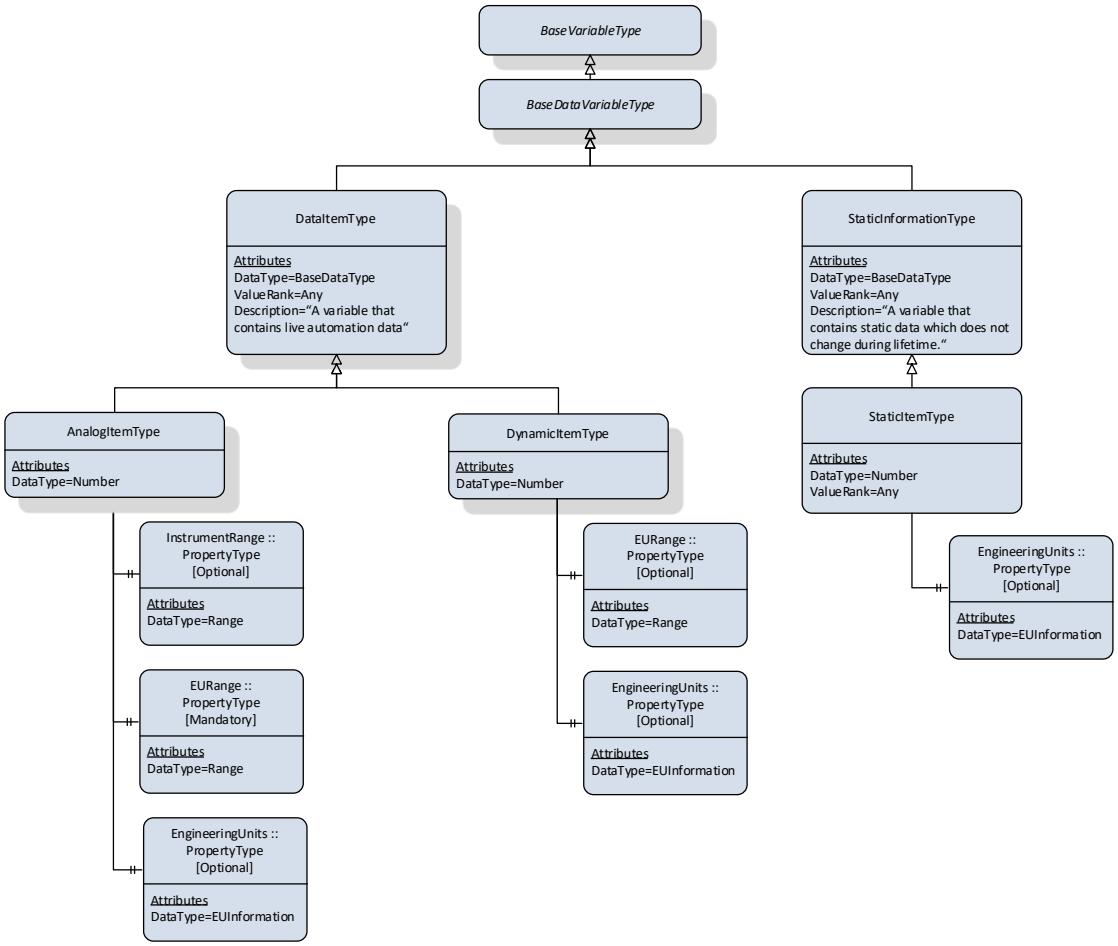

Abbildung 6.6: StaticItemType und DynamicItemType

Eigenheit umgehen können. Dazu wurden zwei Subtypen vom *BaseDataVariableType* eingeführt (siehe Abbildung 6.6). Damit ist eine einheitliche Transformation von UML Attributen in OPC UA Variables möglich.
Der *StaticItemType* wird dazu verwendet, um Daten, welche sich nicht ändern, aber eine Einheit besitzen, zu modellieren. Für Daten ohne Einheit wird der *PropertyType* verwendet.

Für dynamische Daten wird der *DynamicItemType* verwendet. Vielfach wird für Werte, welche sich zur Laufzeit ändern, der *AnalogItemType* verwendet. Der DynamicItem-Type wurde bewusst eingeführt, da es Daten geben kann, die keine Einheit besitzen; somit ist der *AnalogItemType* nicht verwendbar, da bei diesem die Einheiten zwingend zu verwenden sind. Die Unterscheidung erfolgt mit Hilfe des *«AdditionalAttributes».Behavior*.

Wie in Abbildung 6.5 ersichtlich gibt es in UML eine Beziehung zwischen den Klassen

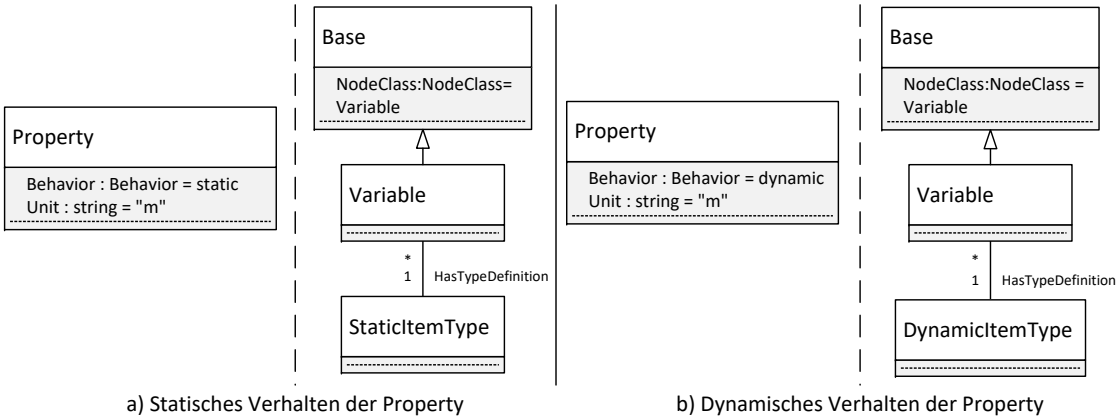

Abbildung 6.7: Mapping von UML Property auf StaticItemType bzw. DynamicItemType

«Class» und *«Property»*. Diese *ownedAttribute* Referenz wird in OPC UA mit einer Referenz zwischen einem *ObjektType* und einer *Variable* dargestellt. Handelt es sich bei der Variablen um eine *Property*, so ist die Referenz *HasProperty* zu verwenden. Bei einer Variablen vom Typ *BaseDataVariable* oder einer ihrer Subtypen wird die Referenz *HasComponent* verwendet.

Folgende Mappingregeln müssen berücksichtigt werden:

- Das Attribut *«Base».NodeClass* wird auf *Variable* gesetzt
- Besitzt das Attribut *«Property».Behavior* keinen Wert, so zeigt die *HasTypeDefinition* auf den PropertyType. Andernfalls treten die nachfolgenden Regeln in Kraft.
- Ist das Attribut *«Property».Behavior* gleich *static* zeigt die *HasTypeDefinition* auf den StaticItemType (siehe Abbildung 6.7 a)
- Ist das Attribut *«Property».Behavior* gleich *dynamic* zeigt die *HasTypeDefinition* auf den DynamicItemType (siehe Abbildung 6.7 b)
- Zeigt die *HasTypeDefintion* auf den *BaseDataVariableType* oder einen seiner Subtypen, so ist eine *HasComponent* Referenz zwischen ObjectType und Variable zu verwenden.
- Verweist die *HasTypeDefinition* auf einen *PropertyType*, so ist eine *HasProperty* Referenz zwischen ObjectType und Variable zu verwenden
- Das Attribut *«Property».Visibility* wird mit dem Attributen *«Variable».AccessLevel* bzw. *«Variable».UserAccessLevel* gemappt. Ist die Visibility gleich private so werden die beiden anderen Attribute auf *CurrentRead* gesetzt, bei *public* gleich *CurrentRead* und *CurrentWrite*. Die beiden anderen Optionen müssen über Object Constraint Language (OCL) Bedingungen abgefangen werden.

Operation2Method

Abbildung 6.8 zeigt die Metamodelle von UML Operations und OPC UA Methods. In UML ist eine *Operation* eine Subklasse von *BehaviouralFeature*. Operations sind Klassen zugeordnet bzw. Klassen können eine oder mehrere Operationen haben. Operationen können wiederum über Input und Output Argumente verfügen.

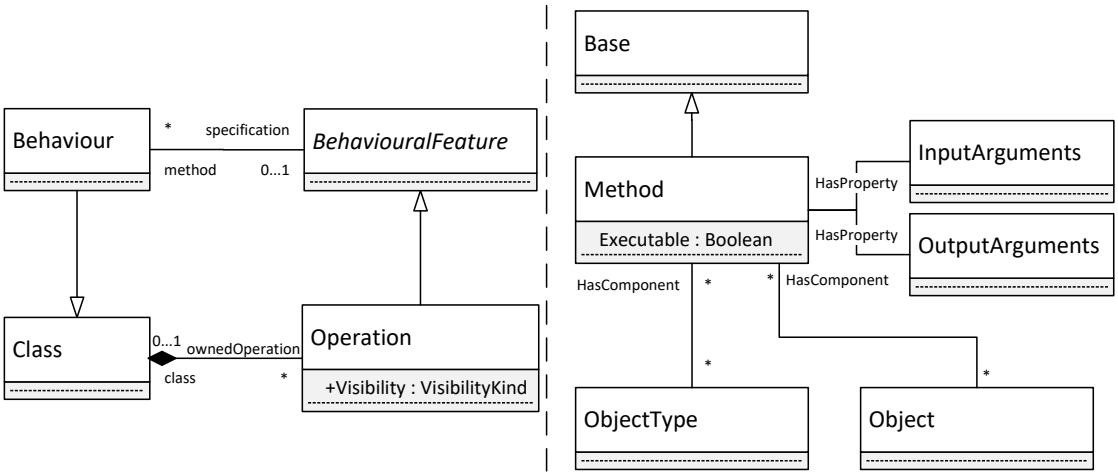

Abbildung 6.8: UML Operation und OPC UA Method

Eine OPC UA Method ist ein Subtyp der Klasse *Base*. Jede Methode besitzt mindestens zwei Properties: Input- und Output-Arguments. Objects und ObjectTypes können OPC UA Methoden über eine HasComponent besitzen.
Somit ergibt sich für das Mapping folgende Regeln:

- Eine Operation wird in eine Methode transformiert.
- Das Attribut «Base».NodeClass wird auf *Method* gesetzt
- Die Input bzw. Output Argumente der Operation werden zu den Input bzw. Output Attributen der OPC UA Methode gemappt
- Die Beziehung *«Class».ownedOperation* wird in eine *HasComponent* Referenz zwischen dem *ObjectType* und der *Method* transformiert
- Das Attribut *«Operation.Visibility»* wird mit dem Attribut *«Method».Executable* gemappt. Ist das Attribut *private* so ist der Wert *false*, bei *public* gleich *true*. Die anderen Varianten von Visibility müssen über OCL Bedingungen abgefangen werden.

DataType2DataType

Abbildung 6.9 zeigt die Metamodelle von OPC UA und UML Datentypen.
In OPC UA ist die Klasse *DataType* eine Subklasse von *Base*. Des Weiteren können Variablen und deren Typen eine *DataType* Referenz auf einen OPC UA Datentyp haben. In UML ist die Klasse *DataType* ein Subtyp von *Classifier*. In UML können sowohl Properties als auch Operationen von einem bestimmten Datentyp sein.

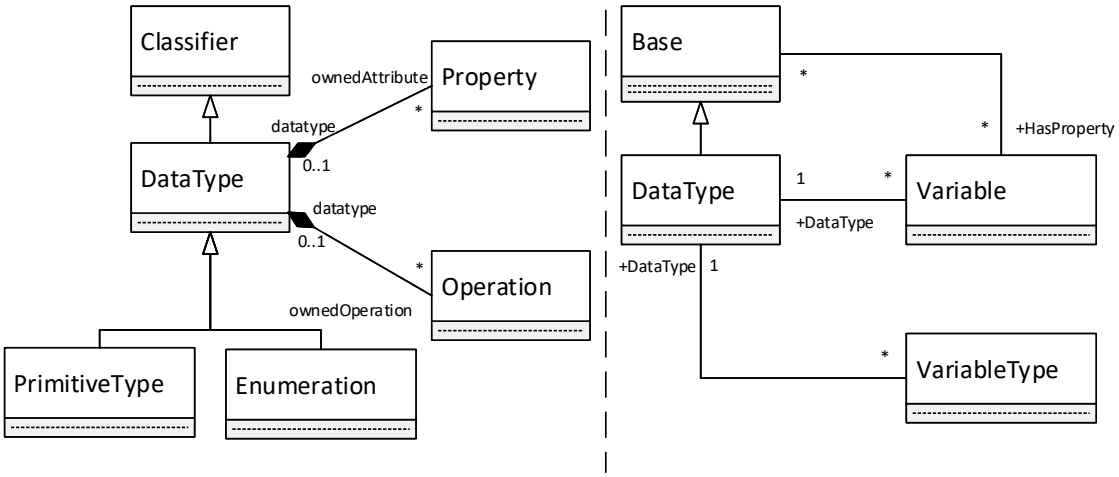

Abbildung 6.9: UML und OPC DataType

Grundsätzlich unterscheidet UML zwischen primitive Datentypen und Enumerations. Die primitiven Datentypen sind (i) Integer, (ii) Boolean, (iii) String und (iv) Unlimited Natural und (v) Real. In OPC UA gibt es jedoch eine Vielzahl von Datentypen. Abbildung 6.10 zeigt die in OPC UA verfügbaren Datentypen.

Für die Transformation von UML zu OPC UA können die primitiven Datentypen in konkrete OPC UA Datentypen umgewandelt werden. Sollten weitere Datentypen benötigt werden, so können diese über das Profil hinzugefügt werden. Das Mapping erfolgt nach folgenden Regeln:

- Der UML Datentyp *Boolean* wird in den OPC UA Datentyp *Boolean* transformiert
- Der UML Datentyp *String* wird in den OPC UA Datentyp *String* transformiert
- Der UML Datentyp *Real* wird in den OPC UA Datentyp *Double* transformiert
- Der UML Datentyp *Integer* wird in den OPC UA Datentyp *Int64* transformiert
- Der UML Datentyp *Enumeration* wird in den OPC UA Datentyp *Enumeration* transformiert

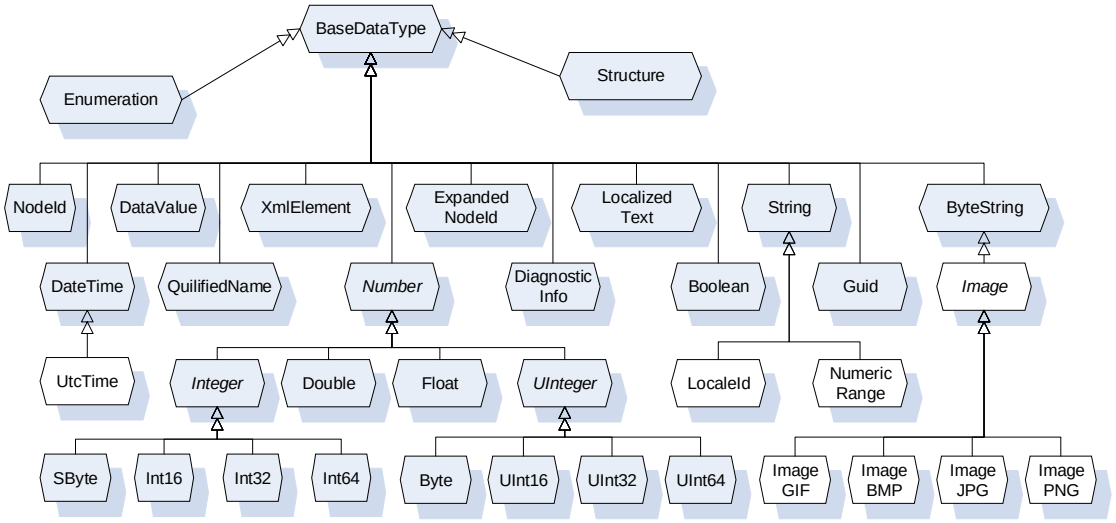

Abbildung 6.10: In OPC UA verfügbare Datentypen

Zusätzlich gibt es eine Einschränkung bei der Modellierung von UML. Laut dem UML Metamodell können *Operations* ebenfalls von einem bestimmten Datentyp sein. Da diese Möglichkeit in OPC UA nicht besteht, muss dies Option mit einer OCL Bedingung abgefangen werden.

Association2Reference

Abbildung 6.11 zeigt das OPC UA Metamodell für Referenzen und Abbildung 6.13 zeigt das entsprechende Metamodell in UML.

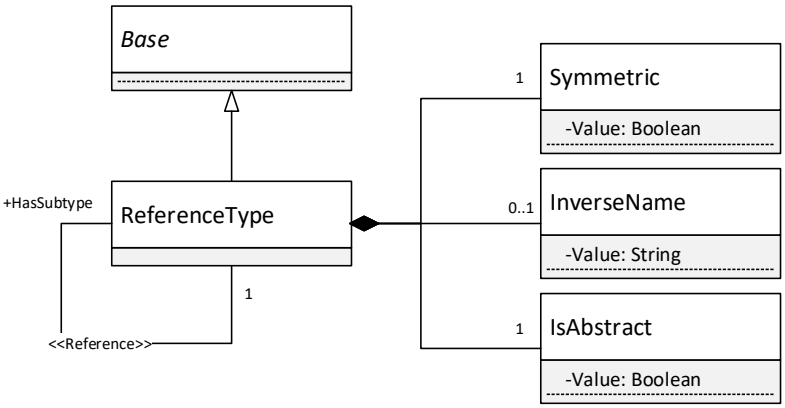

Abbildung 6.11: OPC UA Metamodell References

Der OPC UA *ReferenceType* ist eine Subklasse von *Base*. Das OPC UA Metamodell

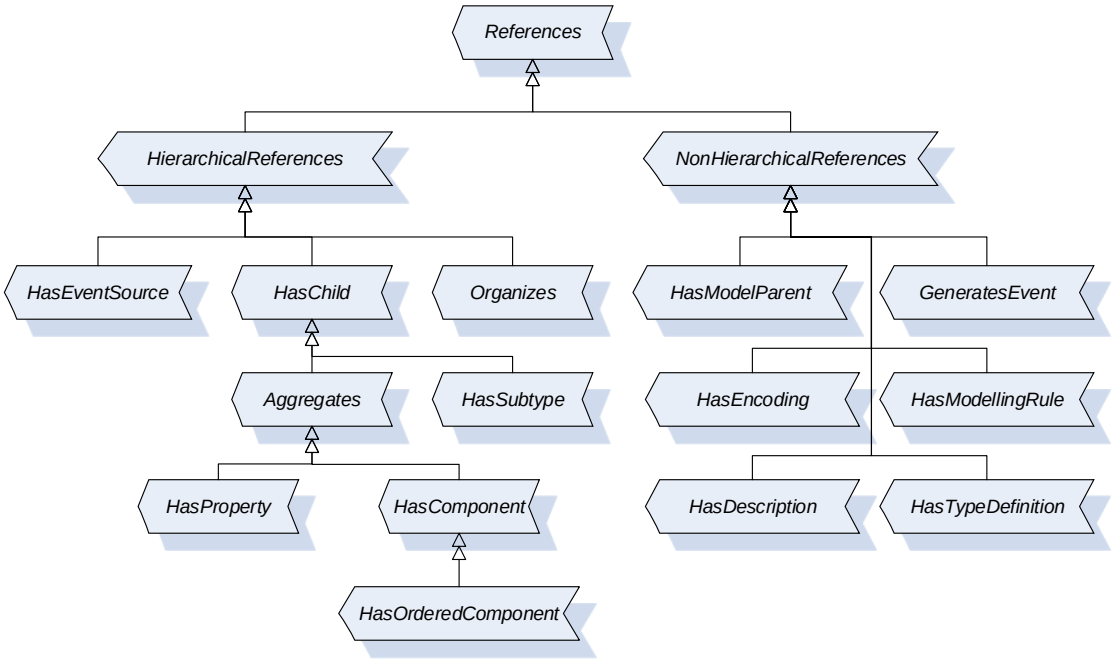

Abbildung 6.12: OPC UA ReferenceTypes [100]

definiert, dass eine Referenz mindestens drei Attribute besitzen muss. Diese sind (i) Symmetric, (ii) InverseName und (iii) IsAbstract. Symmetric beschreibt, ob die Referenz symmetrisch ist, d.h. in beide Richtungen gilt. Das Attribut InverseName definiert die inverse Referenz und IsAbstract gibt an, ob die Referenz abstrakt ist, d.h., nicht instanziiert werden kann.

Referenzen in OPC UA sind hierarchisch mit Subtypen strukturiert. Abbildung 6.12 zeigt die in OPC UA verfügbaren Referenzen. Grundsätzlich werden die Referenzen in *HierarchicalReferences* und *Non-HierarchicalReferences* unterteilt.

UML definiert hingegen nur Generalization und Association. Eine Unterscheidung über die Art der Assoziation wird über das Attribut *aggregation* der Klasse *Property* durchgeführt.

Im Folgenden werden die Mappingregeln zwischen UML und OPC UA Referenzen erklärt:

- Eine UML Generalisierung kann in eine HasSubtype Referenz umgewandelt werden. Die *«Class».superClass* Referenz kann in eine *HasSubtype* Referenz umgewandelt werden.

- Eine Aggregation entspricht einer *Organize* Referenz. In UML spricht man von einer Aggregation, wenn das Attribut *«Property».aggregation* den Wert *shared* hat.

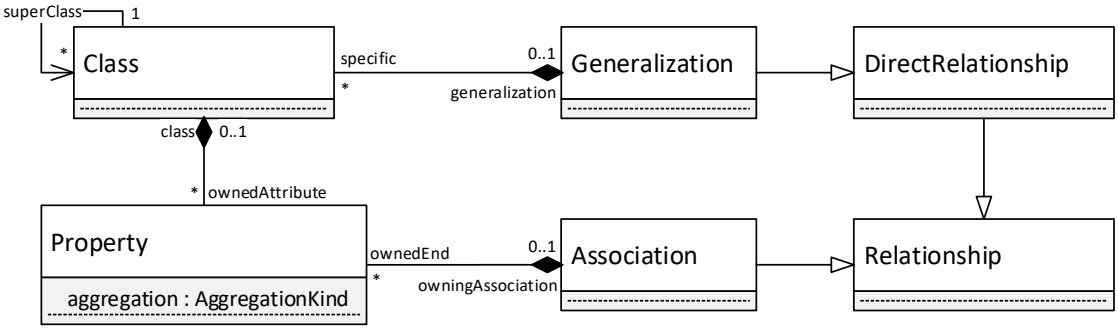

Abbildung 6.13: UML Relationship Metamodell

- Eine Komposition entspricht einer *HasComponent* Referenz. Bei einer Komposition ist der Wert des Attributes *«Property».aggregation* gleich *composite*.
- Eine Assoziation bzw. gerichtete Assoziation entspricht einer nicht hierarchischen Referenz. Bei diesen beiden ist der Wert des Attributes *«Property».aggregation* gleich *none*. Bei einer gerichteten Assoziation wird das Attribut *«ReferenceType».Symmetric* auf *false* gesetzt.

MultiplicityElement2ModellingRule

In UML gibt es das Konzept von Multiplizitäten von Beziehungen. Sie beschreiben die mögliche Anzahl von Instanzen einer Klasse. Abbildung 6.14 zeigt den Ausschnitt des UML Metamodells. UML unterscheidet ab der Spezifikation 2.0 zwischen fünf verschiedenen

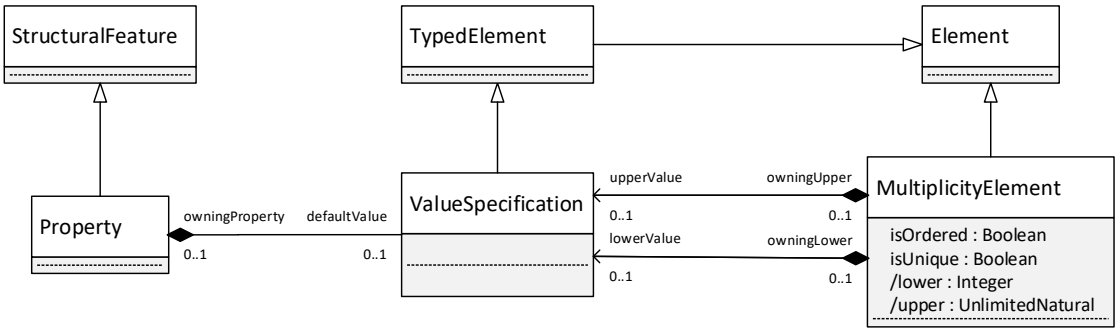

Abbildung 6.14: UML Metamodell von Multiplizitäten

Arten von Multiplizitäten:

1 genau ein Objekt ist an der Beziehung beteiligt

0..1 maximal ein Objekt ist an der Beziehung beteiligt

0..* beliebig viele Objekte sind an der Beziehung beteiligt

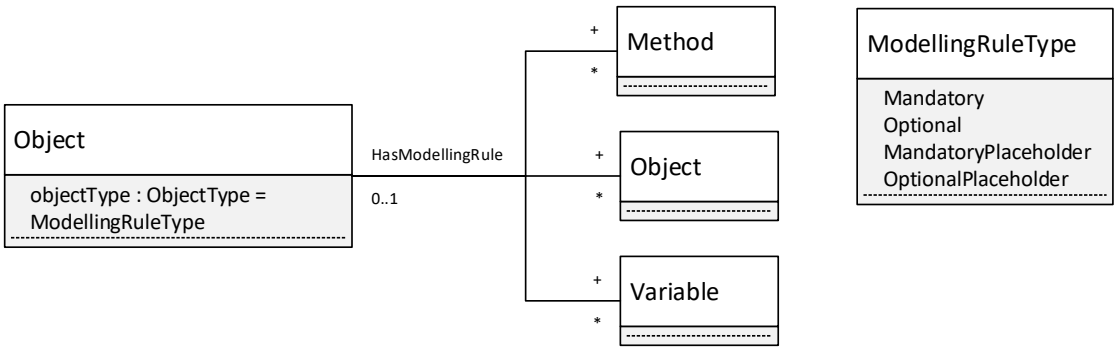

Abbildung 6.15: OPC UA Modelling Rules

1..* mindestens ein Objekt ist an der Beziehung beteiligt

2..4 zwischen zwei und vier Objekte sind an der Beziehung beteiligt

In OPC UA gibt es ein ähnliches Konzept, die *ModellingRule*. Diese gibt ebenfalls an, wie sich Knoten bei der Instanziierung verhalten. Das Metamodell ist in Abbildung 6.15 dargestellt. Die NodeClasses *Object*, *Variable* und *Method* verfügen über eine *HasModellingRule* Referenz. Diese verweist auf ein Objekt vom Typ *ModellingRule*.

Um ein einheitliches Modell zu bekommen, wurden folgende Transformationsregeln definiert. Diese sind in Abbildung 6.15 und Abbildung 6.17 dargestellt.

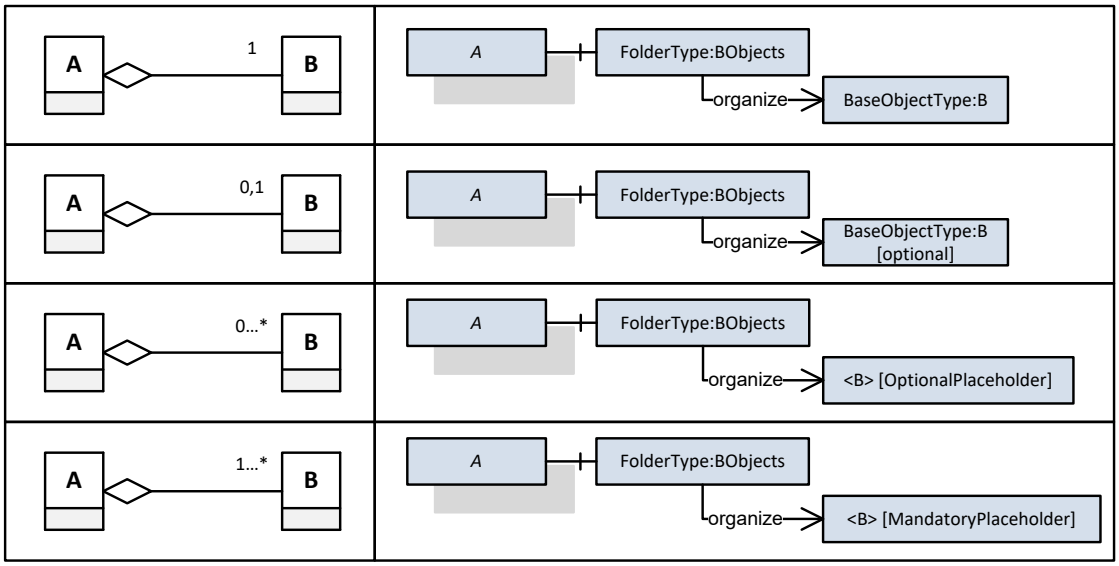

Abbildung 6.16: Transformation einer Aggregationsbeziehung zwischen zwei Klassen

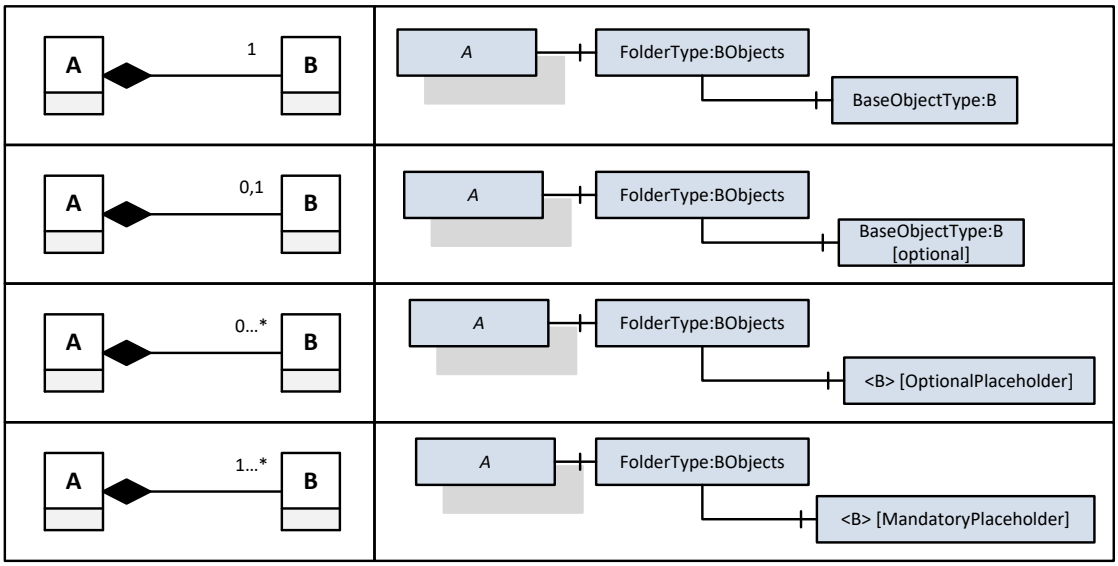

Abbildung 6.17: Transformation einer Kompositionsbeziehung zweier Klassen [114]

all Um den Zugriff auf Objekte im Adressraum möglichst zu vereinheitlichen, wird immer folgende Regel angewandt. Bei einer Beziehung zwischen zwei Klassen *«A»* und *«B»* wird immer ein Objekt vom Typ *FolderType* mit einer *hasComponent* Referenz zum *ObjectType A* erzeugt. Dieser Folder beinhaltet nun die Instanzen von der Klasse *«B»*. Der Name des Folders lautet *«B».name+"Objects"*.

1 Bei Kardianlität 1 wird ein Objekt vom Typ *BType* angelgt. Dieser ObjectType ist mit der Regel Class2ObjectType erstellt worden. Die *ModellingRule* des Objektes *B* ist *mandatory*.

0,1 Hier wird das Attribut *«ModellingRuleType»* des Objektes *B* auf *optional* gesetzt.

0..* Die *ModellingRule* des Objektes *B* wird auf *OptionalPlaceholder* gesetzt

1..* Die *ModellingRule* des Objektes *B* wird auf *MandatoryPlaceholder* gesetzt

Es fällt auf, dass die in OPC UA vorhanden ModellingRules nicht alle Varianten der Kardinalitäten in UML abdecken. Daher müssen folgende Einschränkungen des UML Metamodells getroffen werden:

- Die obere Schranke darf max. 1 sein.
- Die untere Schranke darf entweder 0 oder 1 sein

Eine weitere Einschränkung ergibt sich durch die Tatsache, dass Referenzen in OPC UA Instanzen und keine Typen sind. Bei der Erstellung von Informationsmodellen, also keinen Instanzen, kann die Multiplizität nur bei dem Quellobjekt berücksichtigt werden,

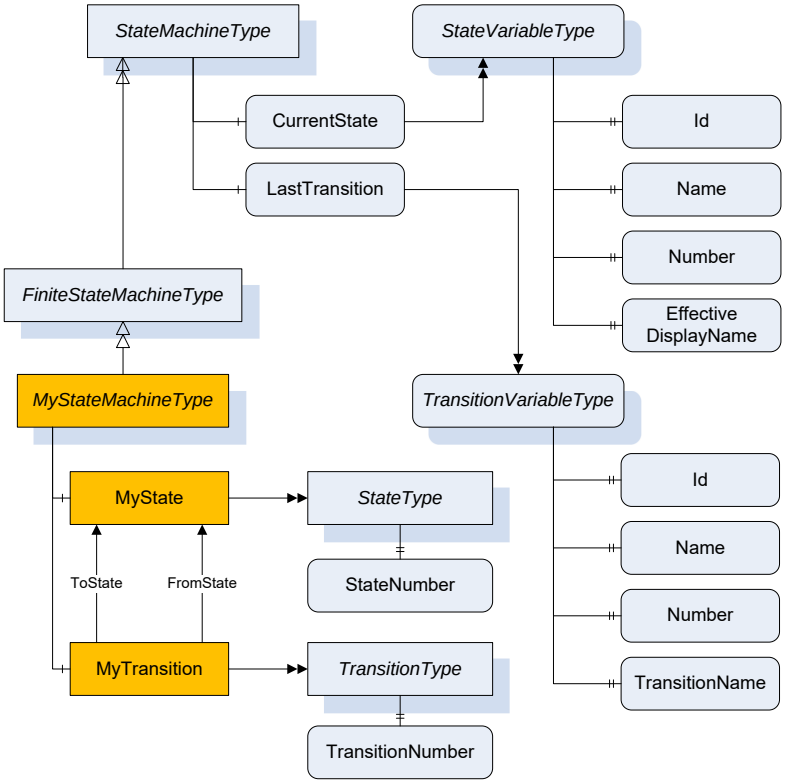

Abbildung 6.18: Beispiel OPC UA StateMachine

da sich diese als ModellingRule manifestiert. Eine Referenz auf mehrere Objekte der Ursprungsklasse ist bei diesem Ansatz nicht möglich. Diese Einschränkung muss ebenfalls mit einer OCL Constraint abgefangen werden.

6.1.2 Zustandsdiagramm2OPCUA

State Machines werden im OPC UA AddressSpace als Objekte abgeleitet vom *StateMachineType* dargestellt. Abbildung 6.18 zeigt ein Beispiel für eine StateMachine in OPC UA. Das State Machine Objekt definiert eine Variable vom Type *StateVariableType*, welche den aktuellen Zustand der Maschine repräsentiert. Eine Instanz des *StateMachineType* soll ein Event generieren, sobald ein Zustandswechsel auftritt. Transitions werden als Objekte vom Typ *TransitionType* definiert. Jede valide Transition besitzt genau eine *FromState* und eine *ToState* Referenz, welche auf ein Objekt vom Typ *StateType* zeigt. StateMachines in OPC UA können auch durch komplexe Funktionen, wie Sub-Statemachines, parallele Zustände, Verzweigungen und Vereinigungen, erweitert werden.

Statemachine2StatemachineType

Eine UML «StateMachine» wird in ein OPC UA StatemachineType als Subtyp des *FiniteStatemachineType* transformiert. Der «Statemachine».name entspricht dem *BrowseName* und *DisplayName* des SubTypes ergänzt mit der Information *StateMachineType*. Abbildung 6.19 zeigt das Mapping.

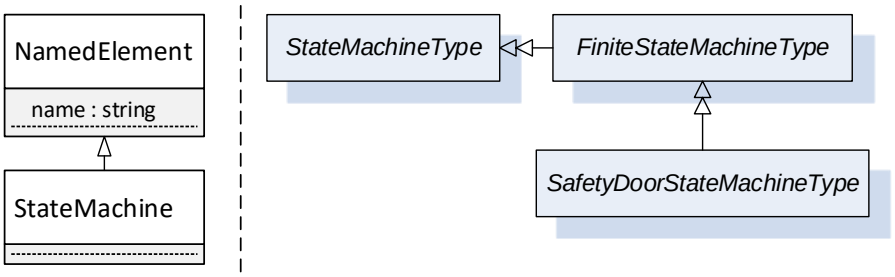

Abbildung 6.19: Transformation Statemachine2Statemachine

State2State

Ein UML.«State» wird in einen Object vom Typ «StateType» transformiert. Abbildung 6.20 zeigt die Zusammenhänge zwischen State und Transition in UML, während Abbildung 6.21 das entsprechende Informationsmodell in OPC UA zeigt. Die Spezialisierung «FinalState» wird in ein Objekt vom Typ «InitialStateType» transformiert.
Der OPCUA.«State».BrowseName und OPCUA.«StateType».DisplayName wird auf UML.«State».name gesetzt. Das Attribut OPCUA.«State».StateNumber ist eine fortlaufende eindeutige Nummer innerhalb einer StateMachine. Diese wird mit einer Helper Funktion während der Transformation erstellt und zugewiesen.

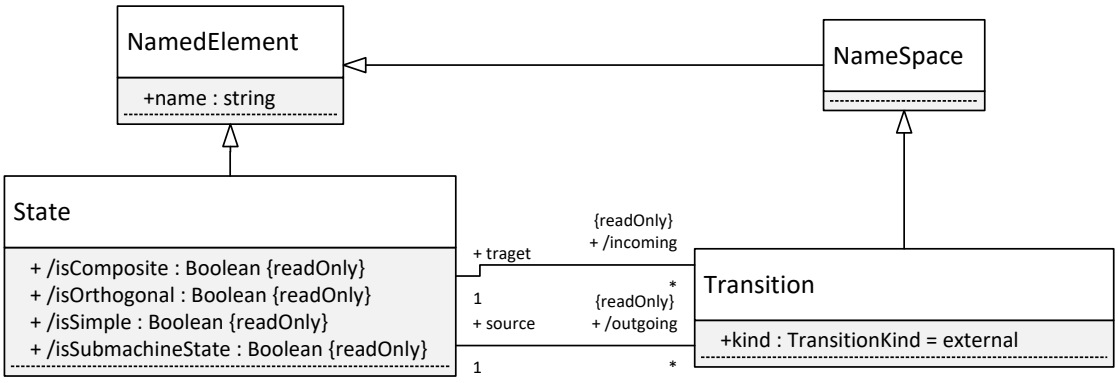

Abbildung 6.20: UML Metamodell von State und Transition

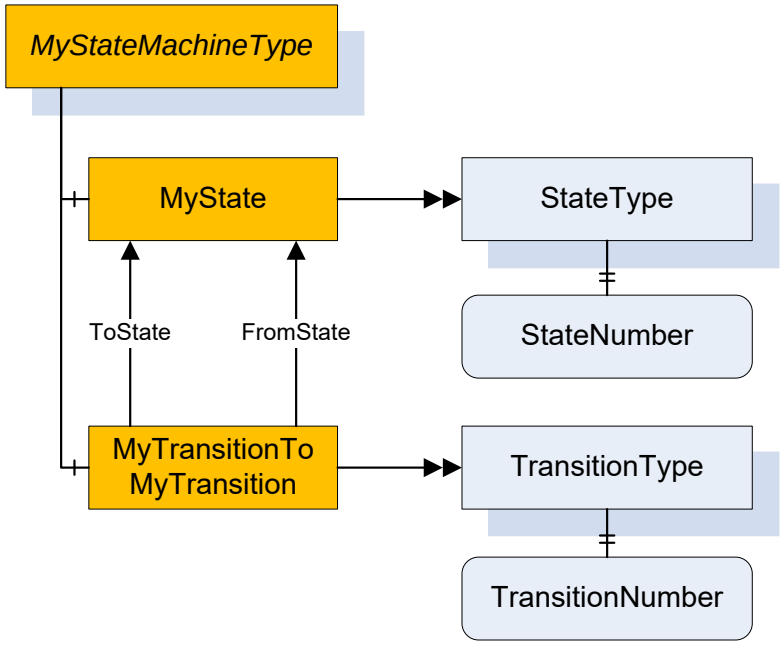

Abbildung 6.21: OPC UA Informationsmodell von Statemachines

Transition2Transition

Eine UML.«Transition» wird in ein OPC UA Objekt MyTransition vom Type *TransitionType* transformiert. Das Attribut OPCUA.«Transition».BrowseName und OPCUA.«Transtion».DisplayName wird auf «source».Name+"To"+«target».name gesetzt.
Die OPCUA.«Transiton».TransitionNumber ist ebenfalls eine fortlaufende eindeutige Nummer, welche die Transitions durchnummeriert. Diese wird analog zur StateNumber mit Hilfe einer Helper Funktion bei der Transformation beschrieben.

Jede Transition in OPC UA muss genau eine *FromState* und eine *ToState* Referenz besitzen, welche auf ein Objekt vom Typ *StateType* zeigt. Die UML.«Transition».target Referenz wird in die *ToState* Referenz transformiert, während die UML.«Transition».source Referenz in die *FromState* Referenz umgewandelt wird.

6.2 Erweiterung von UML mit einem Profil für OPC UA

Das UML Profil, welches speziell für die Transformation von UML Klassendiagrammen entwickelt worden ist, beinhaltet die notwendigen Informationen, um eine eindeutige Transformation von UML zu OPC UA zu gewährleisten. Das Profil ist in Abbildung 6.22 dargestellt.

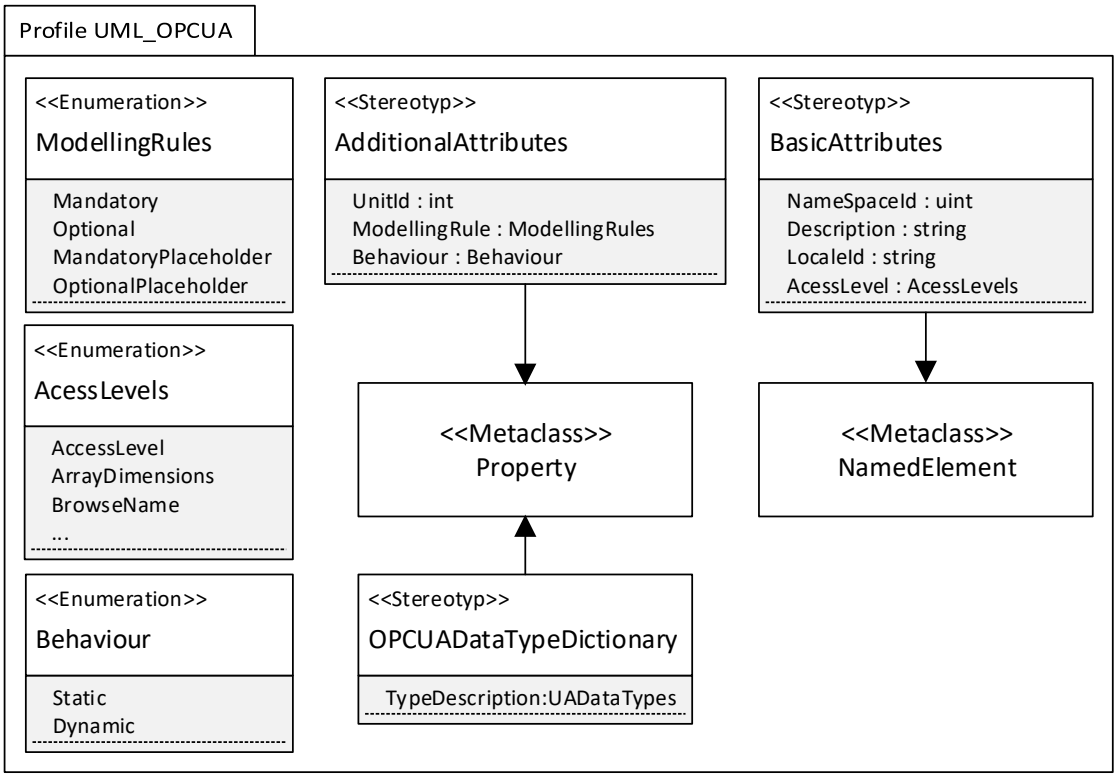

Abbildung 6.22: OPC UA UML Profil [114]

Das Profil ist eine einfache Erweiterung des UML Metamodells. In dem OPCUA Profil für UML werden sowohl Klassen als auch Properties durch zusätzliche Attribute erweitert. Die Eigenschaften der zusätzlichen Stereotypen wird im Folgenden beschrieben:

BasicAttributes: Die wichtigsten Erweiterungen für die korrekte Transformation von UML Elementen in OPC UA ist, dass jeder Knoten (Node) in OPC UA gewisse Attribute benötigt. Konkret muss jede *Node* über eine *NodeId*, einen *BrowseName* sowie einen *DisplayName* verfügen und einem Namensraum zugeordnet sein. Optional kann er auch über eine *Description* verfügen. Dies wird über die Erweiterung der Metaklasse *«NamedElement»* um den Stereotyp *BasicAttributes* erreicht.
Aktuell verfügt die Erweiterung über die Attribute *NameSpaceId*, *Description*, *LocaleId* und *AccessLevel*.

Die *Description* vom Typ *string* ist notwendig, damit jeder Knoten in OPC UA eine Beschreibung erhalten kann.
Die *LocaleId* wird benötigt, um den Datentyp *LocalizedText* in OPC UA darstellen zu können. Somit können Attribute wie z.B. der *DisplayName* in mehreren Sprachen angezeigt werden. Die *LocaleId* ist vom Typ *string* und der Wert muss der ISO 639 [49] entsprechen.
Zuletzt wird jedes Element mit einem *AccessLevel* erweitert. Der *AccessLevel* repräsentiert sowohl die *UserWriteMask* als auch die *WriteMask* des Knotens. Die Literale der Enumeration sind nicht vollständig dargestellt. Eine komplette Liste kann aus der OPC UA Spezifikation Teil 3 [100] entnommen werden.

AdditionalAttributes: Zusätzlich verfügt das Profil über eine Erweiterung der *Metaklasse Property*. Diese Erweiterung ermöglicht es, die notwendigen Informationen für Methoden und Variablen in UML zu hinterlegen.
Mit der *UnitId* kann einem Attribut eine Einheit zugewiesen werden. Die *UnitId* ist in OPC UA eindeutig und somit können weitere Informationen wie z.B. der Name der Einheit herausgefunden werden.
Mit *Behavior* ist das Verhalten der Property gemeint. In OPC UA ist es von Bedeutung, ob sich die Werte eines Datenpunktes zur Laufzeit ändern. Davon abhängig müssen spezielle Datentypen für die Modellierung der Attribute gewählt werden. Eine genaue Beschreibung, wie sich das Attribut *Behaviour* auf die Modelle auswirkt, befindet sich in Kapitel 6.1.
Das dritte Attribut ist die *ModellingRule*. Sie entspricht der OPC UA *ModellingRule*. Da die *ModellingRule* von Objekten über die Multiplizitäten zwischen den Klassen abgebildet werden kann, ist diese Erweiterung nur für UML Attribute notwendig.

OPCUADataTypeDictionary Sinnvollerweise verfügt das Profil über alle Datentypen, welche von OPC UA unterstützt werden. Somit kann einerseits bei der Modellierung der UML Modelle schon auf die Eigenheiten der Technologie eingegangen werden (R-PIM Gedanke) und andererseits bei der Transformation überprüft werden, ob nur Datentypen verwendet werden, welche auch von OPC UA unterstützt werden. Eine Übersicht über alle OPC UA Datentypen ist in Abbildung 6.10 gegeben. Im Profil wird das Attribut *TypeDescription* verwendet. Mit diesem kann über das OPC UA *DataTypeDictionary* auf den Datentyp rückgeschlossen werden. Eine Beschreibung dazu findet sich in der OPC UA Spezifikation Teil 3 [100].

6.3 Erweiterung von OPC UA

Eines der wichtigsten Konzepte von OPC UA ist die *Extensibility*, d.h., das Metamodell kann beliebig erweitert werden. Diese Eigenschaft erlaubt es nun, das OPC UA Metamodell um Funktionen von UML zu erweitern. Dieses Konzept wurde bei den Zustandsdiagrammen (Statemachines) angewendet. Die Ergebnisse, die in dem folgenden Abschnitt präsentiert werden, sind in der Publikation *Guarded State Machines in OPC UA* [33] genau beschrieben.

Genauso wie in UML gibt es auch in OPC UA das Konzept von Statemachines. Diese unterscheiden sich jedoch etwas von den UML Zustandsdiagrammen. Ein *State* ist ein Zustand, in dem sich ein Objekt für eine definierte Zeit befindet. Eine *Transition* ist ein Wechsel von einem Zustand in einen anderen. Dieser Zustandswechsel wird durch ein Ereignis ausgelöst. Im Einklang mit dem Informationsmodell Konzept bedeutet dies, dass jedes Ereignis durch eine Methode repräsentiert wird, welche aufgerufen werden muss, um ein *Event* auszulösen. Grundsätzlich können auch Transitions eine Aktion auslösen. Dies wird *Effect* genannt und ausgelöst, bevor das Objekt den Ziel-Zustand einnimmt.

Im Gegensatz zu UML gibt es in OPC UA keine Möglichkeit, *Constraints* mit Hilfe von *Guards* zu definieren. Diese müssen erfüllt sein, damit eine Transition zwischen zwei Zuständen erfolgen kann. *Guards* können unterschiedlich komplexe Formen annehmen. Von einfachen booleschen Werten bis hin zu Referenzen auf andere Knoten im OPC UA Adressraum und Zustände sind möglich. Um diese Möglichkeiten korrekt in OPC UA abbilden zu können, wurden neue Objekttypen und Referenzen in OPC UA definiert.

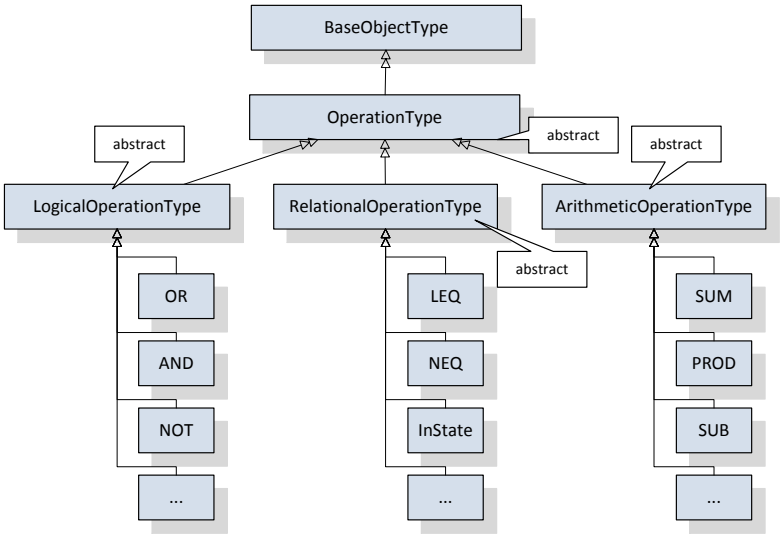

Abbildung 6.23: OPC UA Operation Type

Abbildung 6.23 zeigt den neu eingeführten *OperationType*. Dieser erweitert das OPC UA Metamodell um die Fähigkeit, Knoten mit einem Operanden miteinander zu verknüpfen. Dabei wurden zunächst drei abstrakte Subtypen definiert, welche wiederum einige Subtypen besitzen. Aktuell sind *LogicalOperationType*, *RelationalOperationType*, *ArithmeticOperationType* vorgesehen. Die so definierten Typen erlauben es, komplexe Bedingungen zu definieren. Eine Vollständigkeit ist nicht gegeben, die Typen können bei Bedarf jederzeit erweitert werden.

Zusätzlich wurden noch eigene Referenzen definiert, um Objekten Operationen zuweisen zu können. Abbildung 6.24 stellt die beiden neu eingeführten Referenztypen dar. Die *HasGuard* Referenz zeigt von einer Transition zu einem Guard Objekt, welches die Bedingung für eine erfolgreichen Übergang definiert. Sie ist ein Subtyp des abstrakten *NonHierarchicalReferences* Typs und kann somit auch für Loop Verweise verwendet werden. Die *OperandReferences* ist ein Subtyp der *HierarchicalReferences* und ebenfalls abstrakt. Sie kann dazu verwendet werden, um die Guard Bedingung für einen erfolgreichen Zustandswechsel zu definieren. Dabei werden Knoten mit einer Referenz vom Typ *HasOperand* mit dem Guard verknüpft.

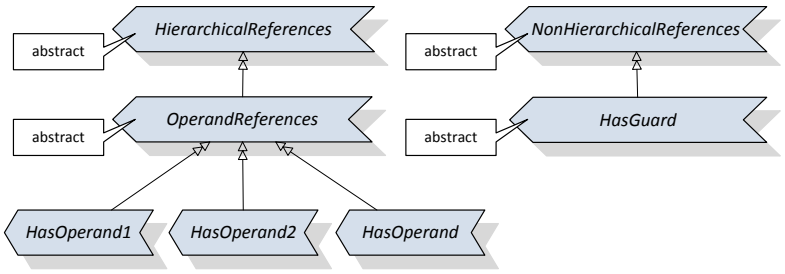

Abbildung 6.24: OPC UA OperandReferences und HasGuard Referenztypen [33]

Wie die neu eingeführten Objekttypen und Referenzen verwendet werden, ist in Abbildung 6.25 dargestellt.

Genau wie in UML ist es auch in OPC UA möglich, die interne Struktur von Zuständen näher zu spezifizieren. Dies gelingt durch die Verkettung mit einer anderen Zustandsmaschine (z.B. *OpenInternalStateMachine*) mit der *HasComponent* Referenz. Als Beispiel soll hier nur der Zustand *Open* diskutiert werden.
Die innere Struktur des Zustandes *Open* und seine Verbindung zur *SafetyDoorStateMachine* ist im oberen Teil von Abbildung 6.25 dargestellt. Der untere Teil veranschaulicht eine sehr vereinfachte Sicht auf den Arbeitsraum einer Maschine. Die WorkingSpaceStateMachine hat zwei mögliche Zustände (Gesperrt und Entsperrt) sowie Übergänge zwischen ihnen und eine Variable, die ihren aktuellen Zustand anzeigt.

Die OpenInternalStateMachine besteht aus der *CurrentState* Zustandsvariable und zwei Zuständen, nämlich *OpenIdle* und *OpenCloseRequested*. Übergänge sind zwischen den beiden Zuständen und von *OpenIdle* zu *OpenCloseRequested* möglich.

Angenommen, der aktuelle Zustand ist *OpenIdle* und in OPC UA würde ein Methodenaufruf an *SafetyDoorClose* erfolgen. Dieser würde entweder einen Übergang in den Zustand *OpenIdle* selbst, der ein *InvalidCloseEvent* erzeugt, oder in den Zustand *OpenCloseRequested*, abhängig vom Code, der diese Entscheidung implementiert, auslösen.

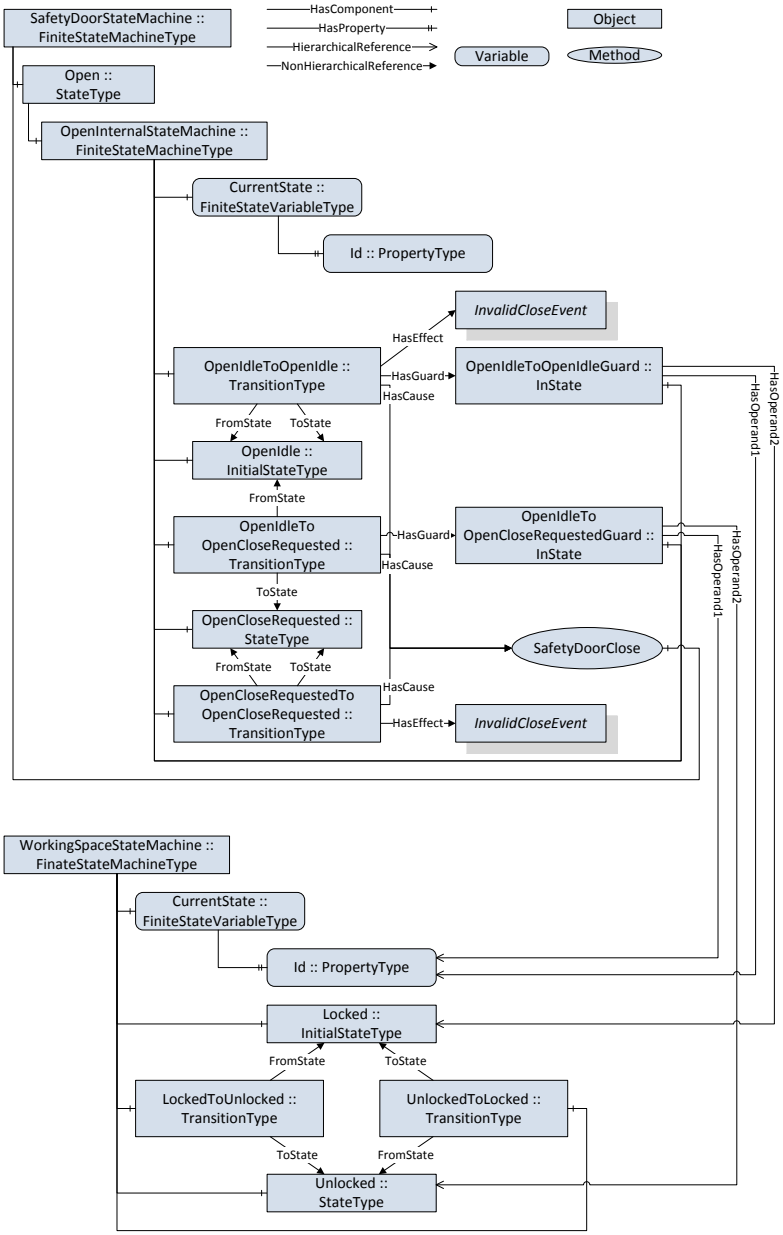

Abbildung 6.25: Beispiel eines OPC UA Guarded State Modells einer Sicherheitstür [33]

Diese Bedingung ist normalerweise für den verbundenen Client beim Ausführen der *SafetyDoorClose* Methode nicht sichtbar.
Eine mögliche Lösung für dieses Problem ist das Hinzufügen eines Objekts des neu definierten Typs *InState* zum Übergang über eine *HasGuard* Referenz. Hier werden die Bedingungen, die erfüllt sein müssen, um einem Übergang auszulösen, im Adressraum exponiert, so dass der *Client* einen Einblick in die Entscheidungen, die zu einem Zus-

tandsübergang führen, erhält. Im vorliegenden Beispiel *WorkingSpaceStateMachine* muss sich die Statemachine im Zustand *Unlocked* befinden, um den Übergang von *OpenIdle* zu *OpenCloseRequested* zu ermöglichen.

6.4 Einschränkung von UML - Constraints

Die Constraints werden dazu verwendet, um den Funktionsumfang von UML einzuschränken. Dies wird immer dann angewendet, wenn das Metamodell von OPC UA nicht erweitert werden kann. Im Folgenden werden konkrete Beispiele für diese Restriktion angegeben:

Das beste Beispiel für Restriktionen sind Mehrfachvererbungen. In UML ist dieses Konzept gängig, d.h., eine Klasse kann Eigenschaften von mehr als einer Superklasse vererbt bekommen. In OPC UA ist dies nicht möglich. Es gibt zwar das Konzept der Vererbung, jedoch darf ein Knoten nur von einem Knoten erben. Als Beispiel können OPC UA Referenzen genannt werden. In OPC UA darf jeder ReferenceType nur genau einen Supertyp besitzen. Die entsprechende OCL Constraint lautet:

```
1 context ReferenceType
2 inv: self.browseName=Reference implies self.superTypes -> size() == 1
```

Der entsprechende OCL Ausdruck, um die UML Modelle einzuschränken, damit sie keine Mehrfachvererbung verwenden, lautet:

```
1 context Class
2 inv: self.generalizations.size() == 1
```

Ein weiteres Beispiel für Einschränkungen von UML sind die zu verwendenden Datentypen. Es ist sinnvoll, nur jene Datentypen in UML zuzulassen, welche auch in OPC UA verfügbar sind (siehe Abbildung 6.10).

Die dritte Einschränkung betrifft die Multiplizitäten. In UML gibt es die Möglichkeit, beliebige Multiplizitäten anzugeben. In OPC UA besteht diese Möglichkeit nicht. In OPC UA gibt es nur die vier Möglichkeiten, welche durch die *ModellingRule* definiert werden. Somit können nur 1..1, 0..1, 0..n oder 1..n Beziehungen abgebildet werden.

```
1 context MultiplicityElement
2 inv: self.lower == 0 || self.lower == 1 && self.upper == 0 || self.upper == 1
     || self.upper == 'n'
```

Codeabschnitt 6.2: Constraint Multiplizitäten

Die oben angeführten Beispiele sind nur ein Auszug der möglichen Einschränkungen. Diese sind unter anderem stark vom Stand der Spezifikation abhängig. Da diese ständig aktualisiert und erweitert werden, gelten die oben angeführten Regeln nur für Version 1.3 der OPC UA Spezifikation.

6.5 Transformation zwischen UML und OPC UA

Für die Umsetzung des Mappings wurde Atlas Transformation Language (ATL) verwendet. Nachfolgend wird ein Auszug aus der Transformation präsentiert (siehe auch Codeabschnitt 6.3). Speziell werden hier eine *MatchedRule* und ein *Helper* präsentiert. Diese Elemente stellen die wesentlichen Bestandteile von ATL dar.

```
1 helper context UML!Operation
2 def: getNodeId() : String = 'ns= '+thisModule.namespaceId.get(self.namespace.
      namespace.URI).toString() + ', String, '+ self.name;
3 helper context UML!Class
4 def: getNodeId() : String = 'ns= '+thisModule.namespaceId.get(self.namespace.
      URI).toString() + ', String, '+ self.name+'Type';
5 helper context UML!Enumeration
6 def: getNodeId() : String = 'ns= '+thisModule.namespaceId.get(self.namespace.
      URI).toString() + ', String, '+ self.name;
7 helper context UML!Property def: getNodeId() : String = 'ns= '+thisModule.
      namespaceId.get(self.namespace.namespace.URI).toString() + ', String, '+
      self.name;
8 helper context UML!Parameter
9 def: getNodeId() : String = 'ns= '+thisModule.namespaceId.get(self.namespace.
      namespace.namespace.URI).toString() + ', String, '+ self.name;
```

Codeabschnitt 6.3: Helper für NodeIds

Das wichtigste Konzept in OPC UA ist die *NodeId* . Diese wird bei der Transformation oft verwendet. Da die Ermittlung der *NodeId* in unterschiedlichen ATL Regeln verwendet wird, wurden *Helper* für die *NodeId* definiert. Dadurch lässt sich das Verhalten von unterschiedlichen Regeln an einem Codeabschnitt zusammenfassen. Die Helper Regel *getNodeId* in den unterschiedlichen Kontexten ist in Codeabschnitt 6.3 dargestellt.

Matched Rules sind eines der Kernkonzepte von ATL. Diese Regeln bestehen aus einem *from*-, *to*- und einem optionalen *do*-Block. In dem *from*-Block sind die Elemente des Eingabemetamodells deklariert (Quellmuster) und im *to* Block die Elemente von dem gewünschte Ausgabemetamodell (Zielmuster). Basierend auf diesen Deklarationen wird bei der Ausführung der Transformation geprüft, ob eine Regel für ein gegebenes Eingabeelement ausgeführt wird oder nicht. Die Regel wird dann ausgeführt, wenn das Element mit dem definierten Konzept des Quellenmusters übereinstimmt. *Lazy Rules* sind ähnlich wie *Matched Rules*, müssen aber explizit aufgerufen werden, was für die Zerlegung größerer übereinstimmender Regeln nützlich ist.

```
1 rule Class2ObjectType {
2 from
3 c : UML!Class
4 to
5 ot : OPC!UAObjectType (
6 browseName <- thisModule.namespaceId.get(c.namespace.URI).toString() + ',
      String, '+ c.name + 'Type',
7 displayName <- lt,
8 nodeId <- c.getNodeId(),
```

```
9 isAbstract <- c.isAbstract,
10 references <- lor
11 ),
12 lt : OPC!LocalizedText(
13 value <- c.name+'Type'
14 ),
15 lor : OPC!ListOfReferences(
16 reference <- c.superClass -> collect(s|thisModule.generateSubTypeRel(s)),
17 reference <- c.ownedOperation -> collect(s|thisModule.generateOperations(s)),
18 reference <- c.ownedAttribute -> collect(s|thisModule.generateAttributes(s))
19 )
20 }
```

Codeabschnitt 6.4: Matched Rule für Class2ObjectType

Codeabschnitt 6.4 zeigt die *Matched Rule* für die Transformation von Klassen in Objekttypen. Die Zeile 1 bis 5 definieren die Regel. Es wird eine UML Klasse in einen OPC UA *ObjectType* transformiert. In den Zeilen 6 bis 10 erfolgt die Zuweisung der Attribute. Zeile 8 ruft die Lazy Rule *getNodeId* auf, welche die NodeId des *ObjectTpyes* bestimmt. In den Zeilen 12 und 13 wird der Displayname als *LocalizedText* ermittelt. In den Zeilen 15 bis 18 werden alle Referenzen der Klasse ermittelt und die daran hängenden Objekte erstellt.

```
1 lazy rule generateOperations{
2 from
3 o : UML!Operation
4 to
5 r : OPC!Reference (
6 referenceType <- 'HasComponent',
7 value <- o.getNodeId()
8 )
9 }
```

Codeabschnitt 6.5: Lazy Rule für das Generieren von Methoden

```
1 lazy rule generateSubTypeRel{
2 from
3 c : UML!Class
4 to
5 r : OPC!Reference (
6 referenceType <- 'HasSubtype',
7 isForward <- false,
8 value <- thisModule.resolveTemp(c, 'ot').nodeId
9 )
10 do{r;}
11
12 }
```

Codeabschnitt 6.6: Lazy Rule für das bestimmen der Subtypen

Codeausschnitt 6.5 zeigt den Code, der die erforderlichen Methoden generiert. Codeabschnitt 6.6 zeigt die Lazy Rule, welche für die Generierung der Subtypen verantwortlich ist. Sie transformiert die *superClass* Referenz von UML in eine OPC UA *HasSubtype* Referenz.

KAPITEL 7

Proof of Concept

In diesem Kapitel wird die vorgestellte Vorgehensweise anhand eines Beispiels erläutert. In der Pilotfarbik der TU Wien gibt es unzählige Geräte, welche mittels OPC UA angebunden werden sollen. Für das fahrerlose Transportsystem (FTS), oder auch Automated Guided Vehicle (AGV) genannt, wird das Informationsmodell mit der in Kapitel 5.2 vorgestellten Methodik erstellt. Dadurch kann die Methodik evaluiert werden. Im Folgenden werden schrittweise alle notwendigen Artefakte definiert.

7.1 CIM - Anforderungsanalyse

Zu Beginn wurde die Domäne *mobile Roboter*, im Speziellen *fahrerlose Transportsysteme*, analysiert und die Artefakte entsprechend der Methode erstellt.

7.1.1 Glossar

Dieser Abschnitt beinhaltet eine kurze Beschreibung der wesentlichen Begriffe, welche in den erstellten Dokumenten vorkommen. Dieses Glossar schafft somit ein gemeinsames Verständnis für die Begrifflichkeiten.

AGV-System Die AGV-Systeme bestehen im Wesentlichen aus Fahrzeugen, Peripheriegeräten, Steuergeräten, on-site Komponenten sowie einem stationären Steuerungssystem. Nur das einwandfreie Zusammenspiel all dieser Komponenten sorgt für effizient arbeitende Anlagen.

FTS / AGV Ein FTS oder auch AGV ist ein fahrerloses Radfahrzeug, das automatisch einer Route folgt. Sie sind die zentralen Elemente eines AGVs, da sie die eigentlichen Transportaufgaben ausführen. Die Fahrzeuge müssen individuell nach den spezifischen Gegebenheiten der Umgebung, in der sie eingesetzt werden, gestaltet werden.

Controller Der Controller ist die Steuerungsplattform für alle Funktionen des fahrerlosen Transportsystems wie Navigation, Bewegungsabläufe, Kommunikation, Bedienoberflächen und Sicherheit. Es unterstützt PC- und Benutzerschnittstellen über unterschiedlichste Schnittstellen.

LAM – Lastaufnahmemittel Das Lastaufnahmemittel ist abhängig von der zu transportierenden Last und der Lastübernahme. Es stellte die Hardware Schnittstelle zwischen dem AGV und anderen Systemen dar. Häufig werden Riemen- oder Rollenförderer eingesetzt.

Peripherie Periphäre Systemkomponenten stellen die Gegenstücke zu verschiedenen Bordausrüstungen der Fahrzeuge dar. Beispiele hierfür sind Batterieladestationen und Übergabestationen für das Transportgut.

Auxiliary Darunter werden alle unterstützenden Systeme des AGVs zusammengefasst. Beispiel hierfür ist die Energieversorgung des Systems.

Sensors Sensoren sind die Augen und Ohren eines mobilen Roboters. Neben der Navigation des Systems werden sie auch als Sicherheitssysteme benötigt. Dafür gibt es eine Vielzahl von unterschiedlichen Lösungen.

Roadmap Die Roadmap des AGV beinhaltet eine Landkarte der Umgebung, in der sich das System bewegen soll. Neben den Objekten können hier Zonen, in denen sich das Transportsystem nicht bewegen darf, sowie Bewegungspfade definiert werden.

Subtasks Subtasks sind die generischen Grundfunktionen, zu denen der Roboter fähig ist. Sie beinhalten u.a. das autonome Anfahren von Zielen, das Schalten von Relais und digitalen Ausgängen, auf eine Eingabe warten und eine bestimmte Zeit warten.

Tasks Beliebig viele Subtasks können zu einem Task zusammengefasst werden. Die Aufgabe "Teile von Maschine 1 holen" könnte sich z.B. aus folgenden Teilaufgaben zusammensetzen: (1) Zur Station "Gang1" wechseln, (2) Warnlicht aktivieren, (3) Zur Station "Maschine1" fahren, (4) Lastaufnahme mit dem Lastaufnahmemittel, (5) Zur Station "Gang1" wechseln, (6) Warnlicht deaktivieren.

Sequence Ein Satz von Aufgaben (Tasks) kann dann zu einer kompletten Sequenz zusammengefasst werden. Auf diese Weise können Aufgaben in verschiedenen Sequenzen verwendet werden und müssen nicht immer wieder neu programmiert werden.

7.1.2 Klassendiagramm

In Abbildung 7.1 sind die Zusammenhänge der Begriffe in Form eines einfachen Klassendiagramms dargestellt. Diese bildet den Ausgangspunkt für die Modellierung der statischen Struktur des fahrerlosen Transportsystems und zeigt die Systemgrenzen.

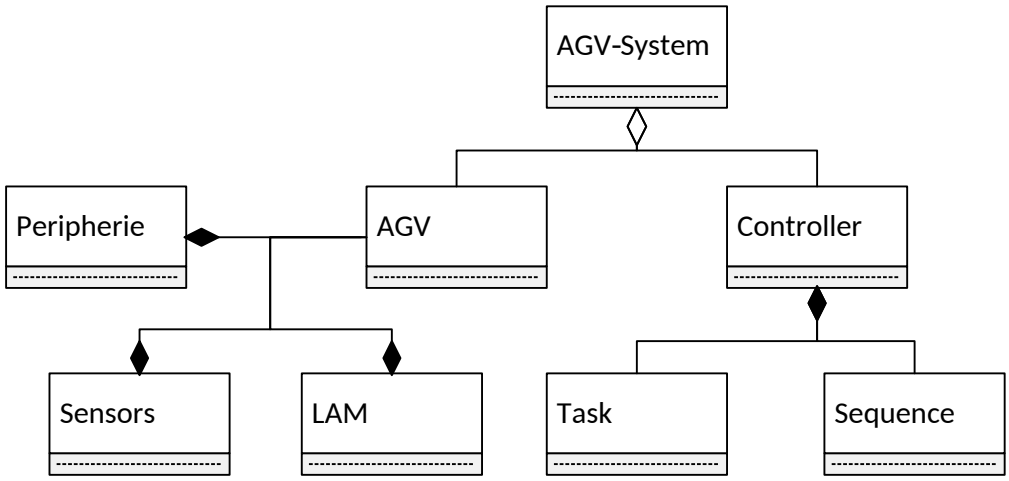

Abbildung 7.1: Zusammenhänge der Begriffe in Form eines Klassendiagramms

Abbildung 7.1 zeigt das AGV im AGV-System. Ein AGV-System besteht nun aus einem AGV und einem Controller. Das AGV ist eine Repräsentation der Hardware, der Controller ist für die Steuerung und Orchestrierung des Fahrzeuges verantwortlich.

7.1.3 Use Case Diagramm

Zur Beschreibung der Funktionen eines AGV wurde ein UseCase Diagramm erstellt. In Abbildung 7.2 ist seine Funktion abgebildet. Folgende UseCases können ausgeführt werden:

Set Task Hier kann der Benutzer oder ein System einen Task anwählen, welcher anschließend ausgeführt werden soll. Dieses Konzept ist aus der Steuerung von Werkzeugmaschinen bekannt. Das AGV hat die Möglichkeit, die Anwahl der neuen Aufgaben abzulehnen.

Execute Task Der Benutzer kann nach der Anwahl einer Aufgabe die Ausführung anstoßen. Das AGV kann auch hier die Abarbeitung ablehnen und in einen definierten Zustand gehen.

Stop Task Während der Abarbeitung eines Tasks kann der Benutzer den Task stoppen.

ResumeTask Ist die Abarbeitung des Tasks gestoppt worden, so kann mit ResumeTask die Abarbeitung wieder aufgenommen werden.

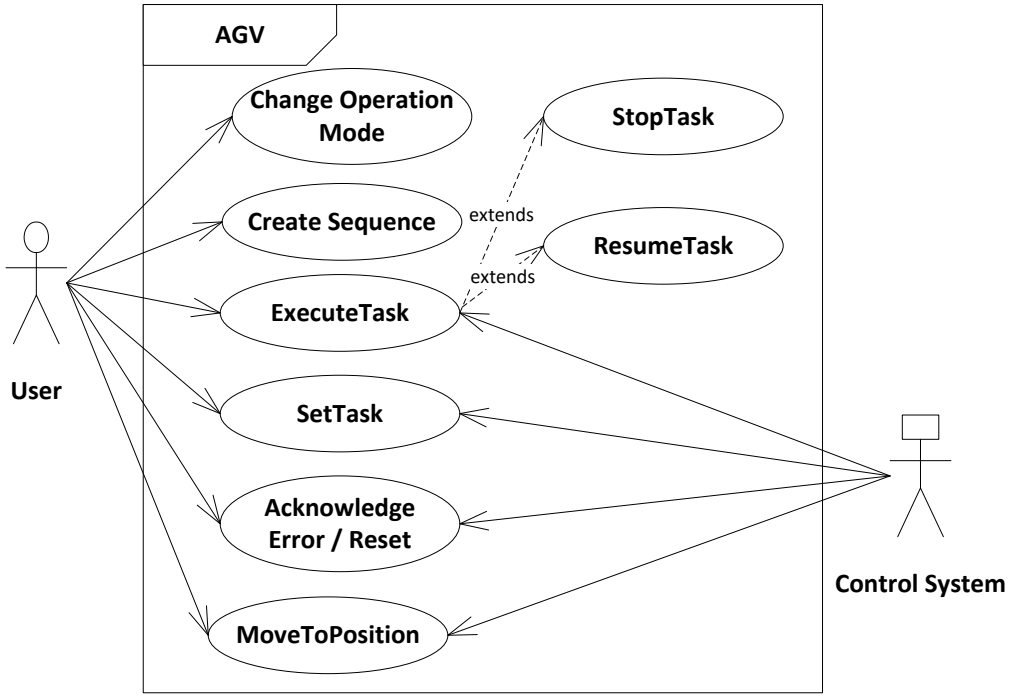

Abbildung 7.2: Vereinfachtes UML UseCase Diagramm für eine Werkzeugmaschine

Acknowledge Error / Reset Ein aufgetretener Fehler bzw. Alarm wird quittiert. Zusätzlich kann das System in einen definierten Zustand gebracht werden. Dies kann entweder durch den Benutzer oder eine übergeordnete Steuerung erfolgen.

Change Operation Mode Der Benutzer kann den Operation Mode des AGVs ändern. Aktuell sind zwei Modi vorgesehen, ein manueller Modus und ein Automatik-Modus. Im manuellen Modus kann der Benutzer das AGV mit dem Joystick steuern. Im Automatik-Modus erfolgt die Steuerung über Tasks, welche von einer übergeordneten Steuerung (Control System) ausgeführt werden.

MoveToPosition Der Benutzer oder ein übergeordnetes System können dem Transportsystem den Befehl schicken sich zu einer zuvor definierten Position zu begeben.

7.1.4 Komponentendiagramm

Abbildung 7.3 zeigt ein vereinfachtes Komponentendiagramm des fahrerlosen Transportsystems Neobotix-MP400 der Pilotfabrik der TU Wien. Es besteht im Wesentlichen aus einem Computer (Controller), einigen Sensoren (Ultraschall und Laserscanner) und zwei Antrieben. Die einzelnen Komponenten sind über unterschiedliche Schnittstellen mit dem Controller bzw. einem Relaisboard verbunden.

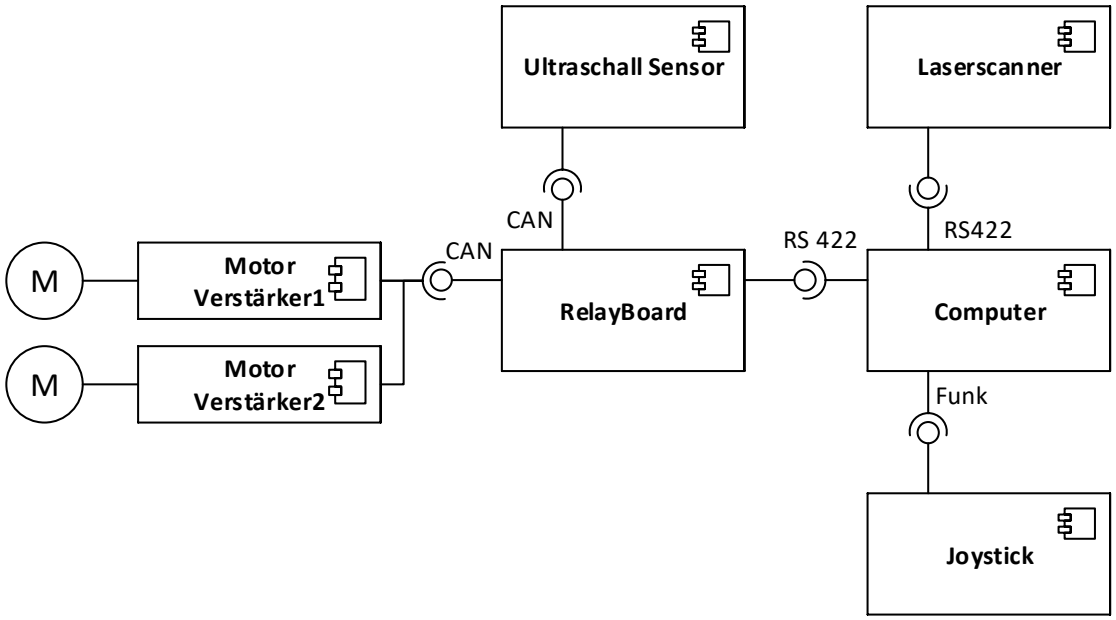

Abbildung 7.3: Komponentendiagramm eines fahrerlosen Transportfahrzeuges [1]

Die Informationen aus dem Diagramm fließen direkt in die Modellierung der statischen Struktur des AGV ein.

7.2 Platform Independent Model (PIM) für AGV

7.2.1 Klassendiagramm

In Abbildung 7.4 ist die statische Struktur eines fahrerlosen Transportsystems abgebildet. Das hier dargestellte Klassendiagramm erhebt keinen Anspruch auf Vollständigkeit. Das erstellte Modell basiert auf dem im CIM entworfenen Klassen- und dem Komponentendiagramm. Ergänzt wurde das Modell um die Attribute aus dem Profil sowie anderer Merkmale, welche die Eigenschaften eines AGV darstellen.
Um eine kompakte Darstellung zu ermöglichen wurde eine angepasste Notation für die zusätzlichen Attribute aus dem UML Profil erstellt.

```
1 Attribute : Type [Unit] {o}
```

In den eckigen Klammern wird die Einheit des Attributes angegeben. Dies entspricht dem Namen der Einheit und muss mit dem UML Profil Attribut *UnitId* abgeglichen werden. Die Angabe ist optional.
In der geschwungenen Klammer wir angeben, ob das Attribut optional oder mandatory

[1] http://www.neobotix-roboter.de/fileadmin/files/downloads/Datenblaetter/MP-400-Datenblatt.pdf

ist. Mit dieser Information wird die *ModellingRule* des Profils dargestellt.
Wird es unterstrichen, so stellt dies dar, dass es statisch ist und stellt somit das *Behaviour* dar.

Das Klassendiagramm in Abbildung 7.4 zeigt die Klassen, welche im CIM Modell definiert und um Attribute und Operationen erweitert wurden.

Ein *AGV* besteht aus den Klassen *Controller*, *Sensors*, *Auxiliary*, und *LoadBearingDevice*. Die Klassen *Sensors*, *Auxiliary* und *LoadBearingDevice* sind abstrakt und können spezialisierte Klassen beinhalten. Das AGV selbst besitzt ein Attribut *OperationMode*, welches eine Enumeration darstellt und über die Methode *ChangeOperationMode* einen Zustandswechsel erlaubt. Die anderen Attribute beschreiben einige Eigenschaften des AGVs.

Die Klasse *Controller* besteht aus der *Roadmap*, *Task*, *Position* und *Sequence* Klasse. Eine Sequenz kann aus mehreren Tasks bestehen. Der Controller beinhaltet neben der Statemachine State die Methoden StartTask, SetTask, Reset und Stop. Diese Methoden wurden schon im UseCase Diagramm im CIM definiert (siehe Abbildung 7.2).
Die Klasse *Position* repräsentiert Positionen, die von dem AGV angefahren werden können. Sie beinhaltet drei Attribute, welche die räumliche Beschreibung der Position erlauben.
Die Klasse *Roadmap* bildet eine Karte des Raumes ab. Sie verfügt über einen Namen und kann optional instantiiert werden. Eine Raumkarte ist nur bei nicht spurgeführten Systemen sinnvoll.
Die Klasse *Sequence* beschreibt Sequenzen aus mehreren Tasks, welche vom AGV ausgeführt werden können.
Die Klasse *Task* beschreibt elementare Aufgaben eines Transportfahrzeugs.

Die Klasse *Auxiliary* ist eine abstrakte Klasse, welche nur ein Attribut *SerialNumber* besitzt. Diese Klasse stellt alle ergänzenden Systeme eines AGVs dar. Aktuell sind drei Spezialisierungen der Klasse verfügbar: die *Battery*, die *DriveUnit* und ein *IOBoard*.
Die Klasse Battery beschreibt die Energieversorgung des Transportfahrzeugs. Sie besitzt Attribute, welche den Ladungszustand und die Grenzwerte für die Spannungsversorgung darstellen.
Mit der Klasse *DriveUnit* wird das Antriebssystem und dessen Antriebsleistung repräsentiert.
Das *IOBoard* stellt eine Ein- und Ausgangskarte dar. Es spezifiziert die Anzahl der Ein- und Ausgänge.

Die abstrakte Klasse *Sensor* bildet die Sensoren des Fahrzeuges ab. Sie besitzt zwei Attribute, eine *Serialnumer* zur Identifikation des Sensors und eine *Status* Enumeration, welche den Status widerspiegelt. Des Weiteren gibt es vier Spezialisierungen dieser Klasse in Form von *Laserscanner*, *Bumper*, *UltrasonicSensor*, *InfraredSensor*. Diese stellen die

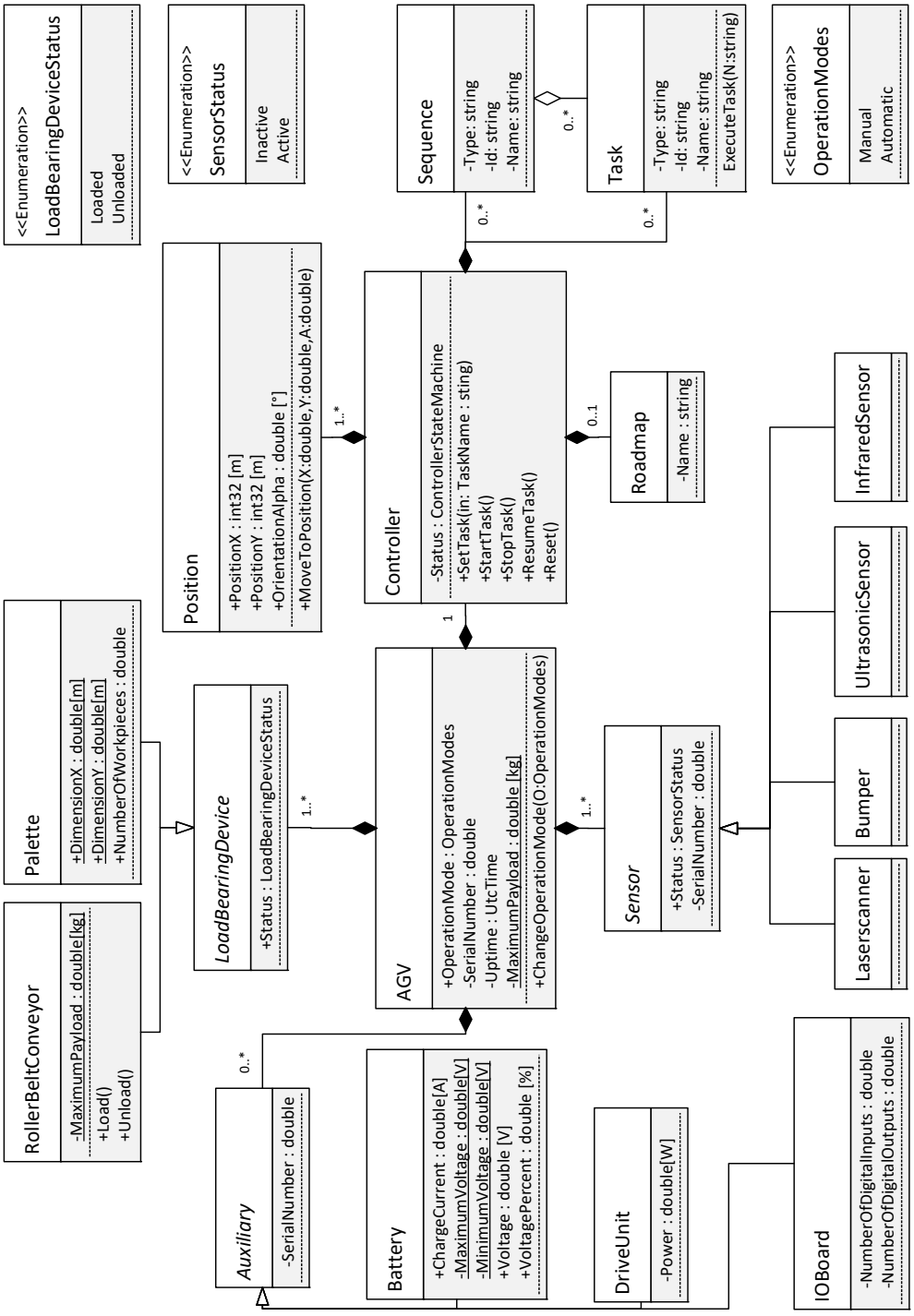

Abbildung 7.4: Klassendiagramm AGV

unterschiedlichen Sensortypen dar.

Die *LadBearingDevice* Klasse stellt die Lastaufnahmemittel des AGV und deren Status dar. Sie bilden die physikalische Schnittstelle zu anderen Systemen. Derzeit gibt es zwei Spezialisierungen: einen *RollerBeltConveyor* und eine *Palette*. Beide besitzen Attribute, welche die Komponenten detaillierter beschreiben.

7.2.2 State Machines für ein AGV

Wie im Klassendiagramm beschrieben gibt es nur eine StateMachine, welche das dynamische Verhalten des AGV beschreibt, die *ControllerStateMachine*. Das UML Diagramm der *ControllerStateMachine* ist in Abbildung 7.5 abgebildet.

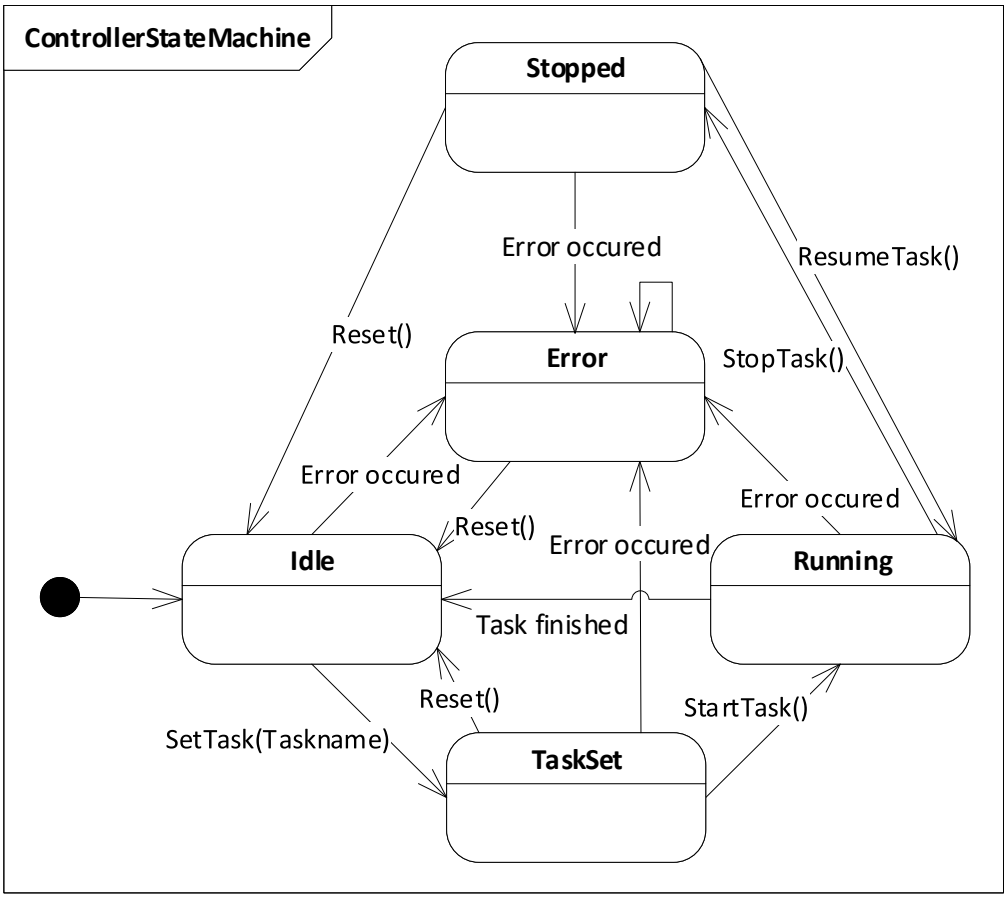

Abbildung 7.5: UML StateMachine Controller State

Zu Beginn nimmt das AGV den Zustand *Idle* ein, indem es so lange verweilt, bis es eine Aufgabe bekommt. Diese wird dem System mit der Methode *SetTask* zugewiesen.

Dadurch erfolgt ein Zustandswechsel in den *TaskSet* Zustand. Von diesem kann mit der Methode *Reset* wieder in den *Idle* Zustand gewechselt werden. Um eine Aufgabe zu starten, wird die *StartTask* Methode aufgerufen. Dadurch wird der Zustand des Systems auf *Running* gesetzt.
In diesem Zustand verweilt es, bis entweder die Methode *StopTask* aufgerufen wird oder das AGV den Task fertiggestellt hat. Bei korrekter Abarbeitung eines Task wird wieder der Zustand *Idle* angenommen. Wird ein Task über die Methode gestoppt, so wird der Zustand *Stopped* eingenommen. Dieser kann nur über die Methode *Reset* in den Zustand *Idle* verlassen werden. Durch Aufruf der Methode *ResumeTask* kann die Abarbeitung einer Aufgabe wieder eingenommen werden und das System wechselt in den *Running* Zustand.
Aus jedem Zustand kann in den *Error* Zustand gewechselt werden. Aus diesem kann mit der Methode *Reset* in den *Idle* Zustand rückgeführt werden.

7.3 OPC UA Informationsmodelle für ein AGV

Abbildung 7.6 zeigt die erste Ebene des hierarchischen Modells, welches eine Übersetzung des UML Klassendiagramms aus Abbildung 7.4 ist. Es zeigt den *AGVType*, welcher die *OperationModeStatemachine*, die Objekte mit Foldern, welche *Sensoren*, *Controller*, *Auxiliary* und *Lastaufnahmemittel* sowie die Variablen *Uptime* und *MaximumPayload*, sowie die Property *SerialNumber* beinhalten. Dies stellt den größten Unterschied zum UML Diagramm dar, da hier neben den Typen Instanzen modelliert werden.
Die nächste Stufe der Detaillierung ist in den folgenden Abbildungen dargestellt. Abbildung 7.7 zeigt den *ControllerType*, Abbildung 7.8 den *SensorType*, Abbildung 7.9 den *AuxiliaryTpye* und Abbildung 7.10 den *SequenceType* und den *TaskType*.

Abbildung 7.11 stellt den Zustandsautomaten des *Controllers* dar und entspricht der Übersetzung des UML Diagramms aus Abbildung 7.5.

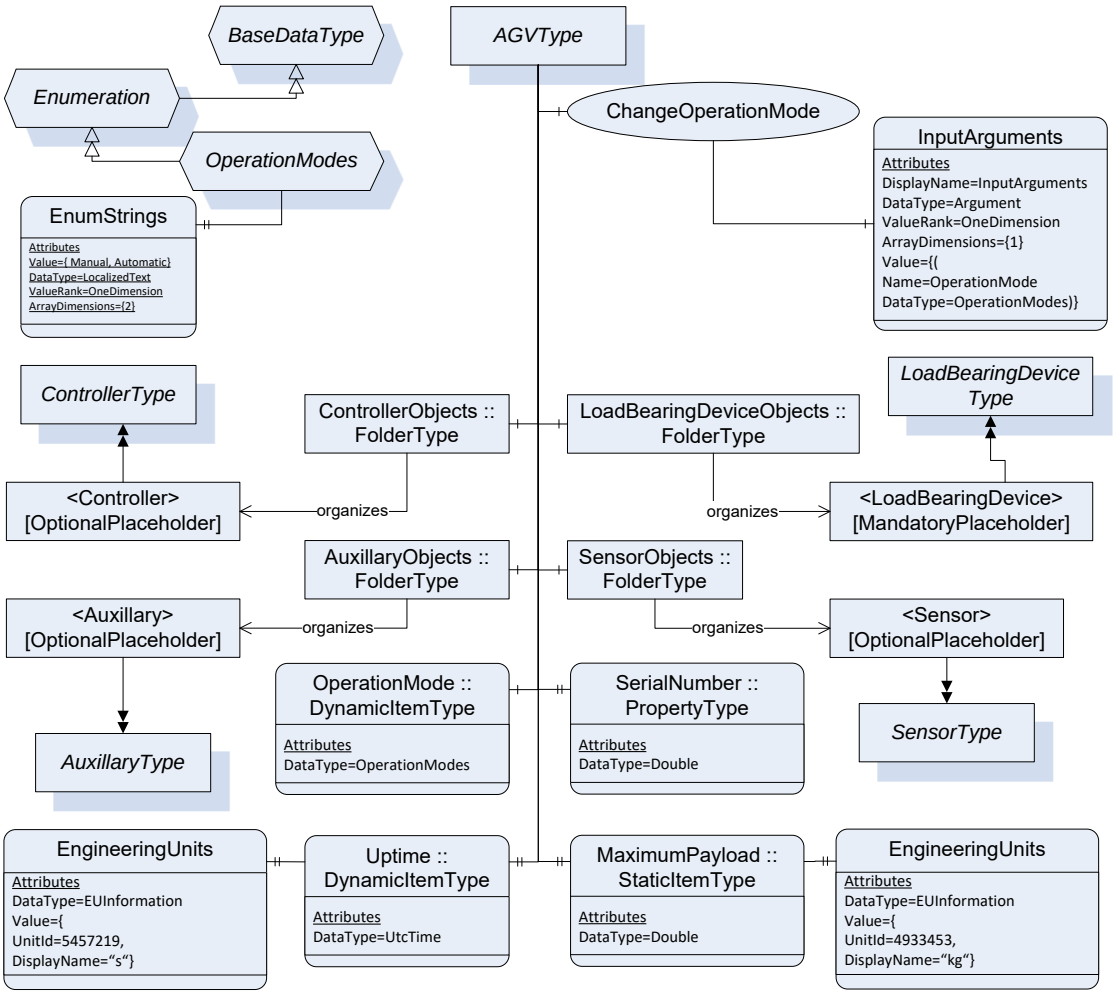

Abbildung 7.6: OPC UA Informationsmodell für das AGV

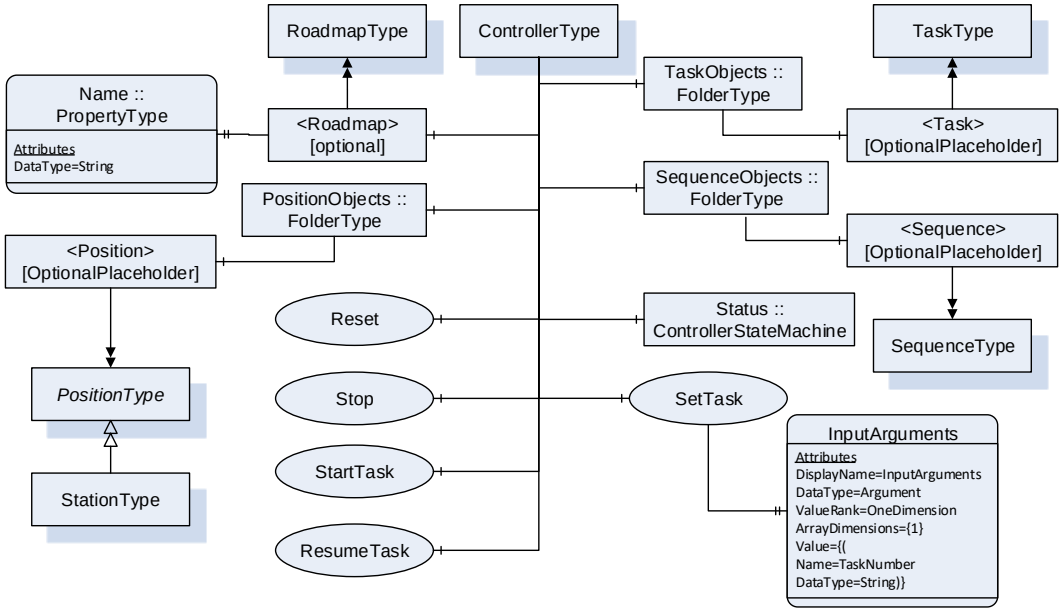

Abbildung 7.7: OPC UA Informationsmodell des Controllers

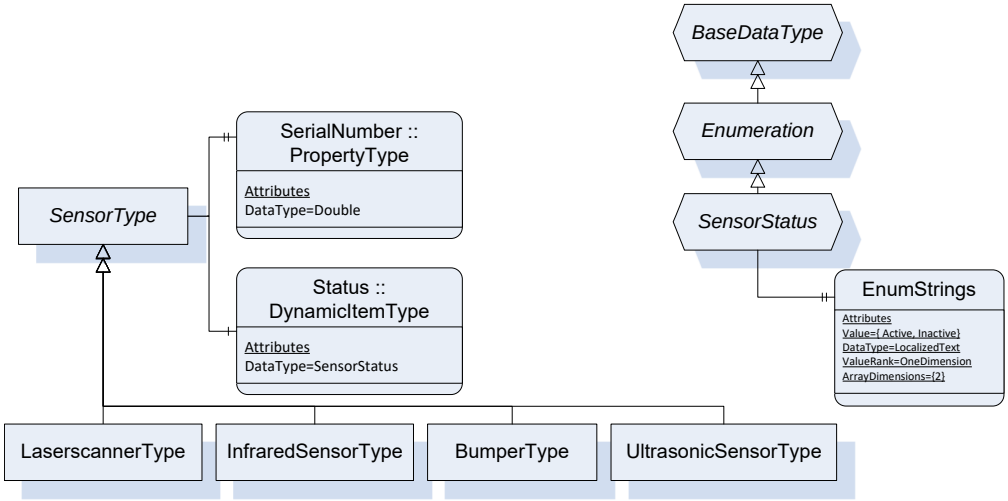

Abbildung 7.8: OPC UA Informationsmodell SensorType

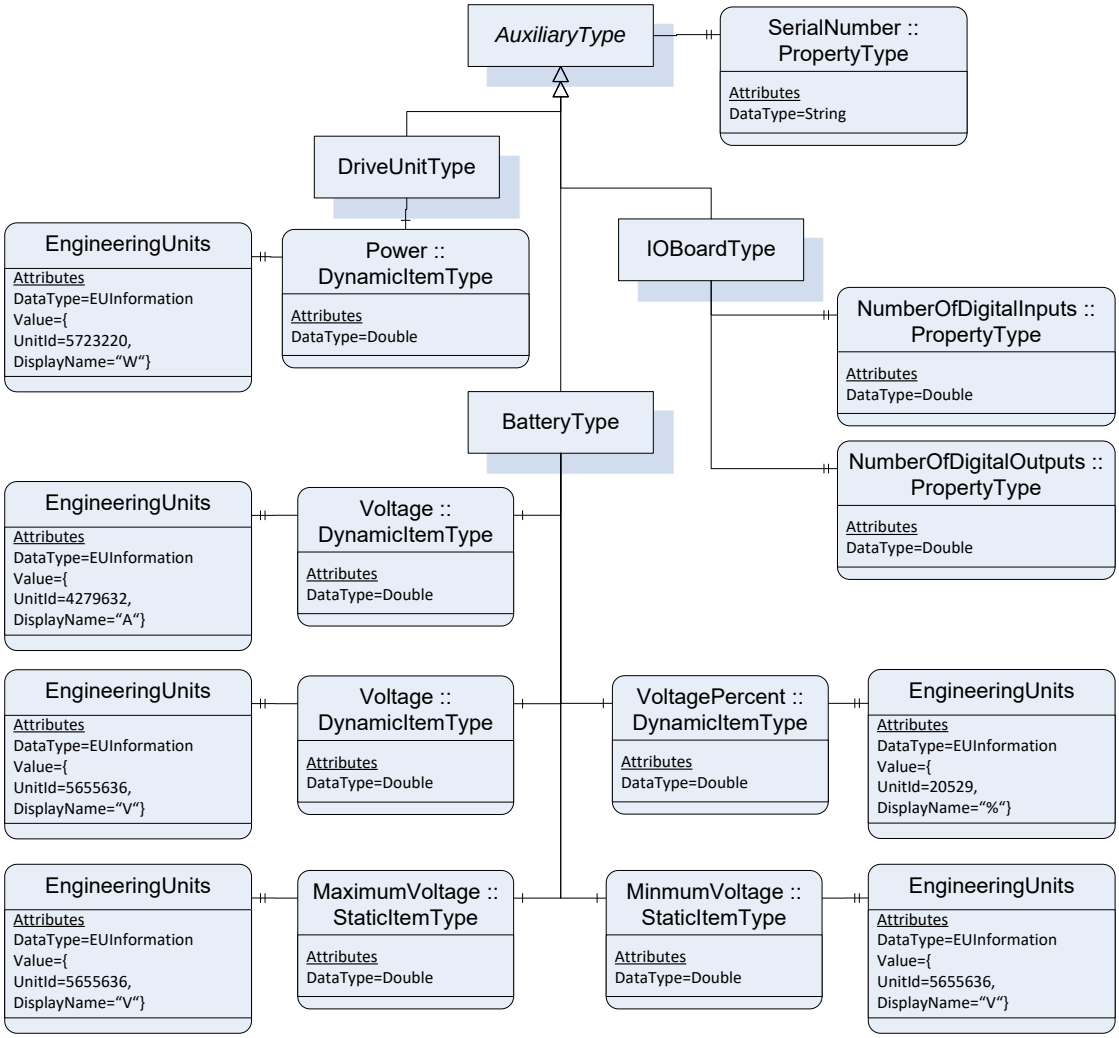

Abbildung 7.9: OPC UA Informationsmodell AuxiliaryType

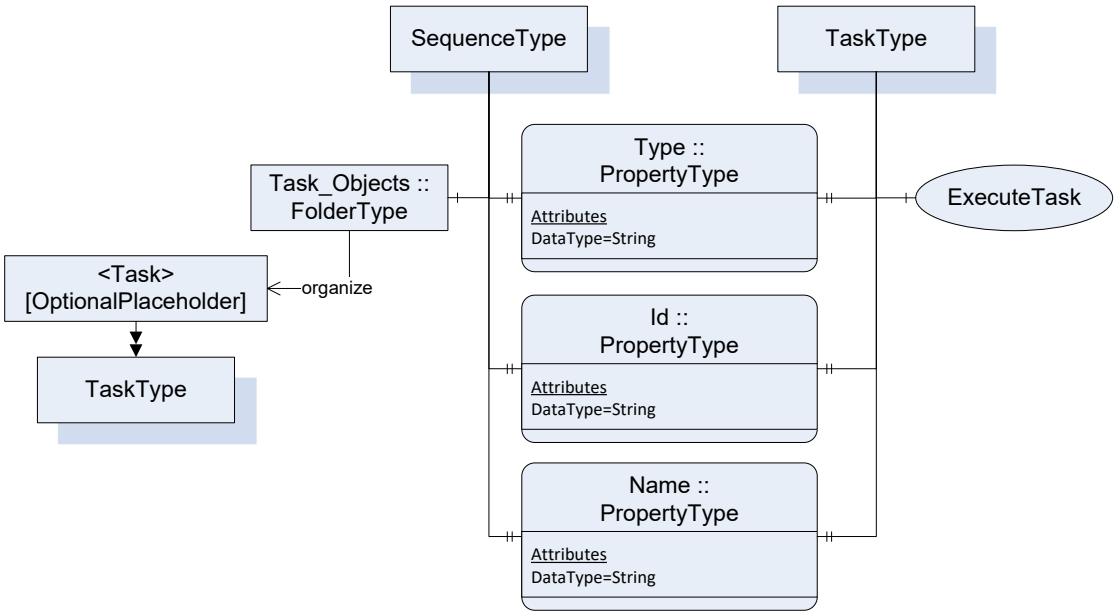

Abbildung 7.10: OPC UA Informationsmodell für den SequenceType und TaskType

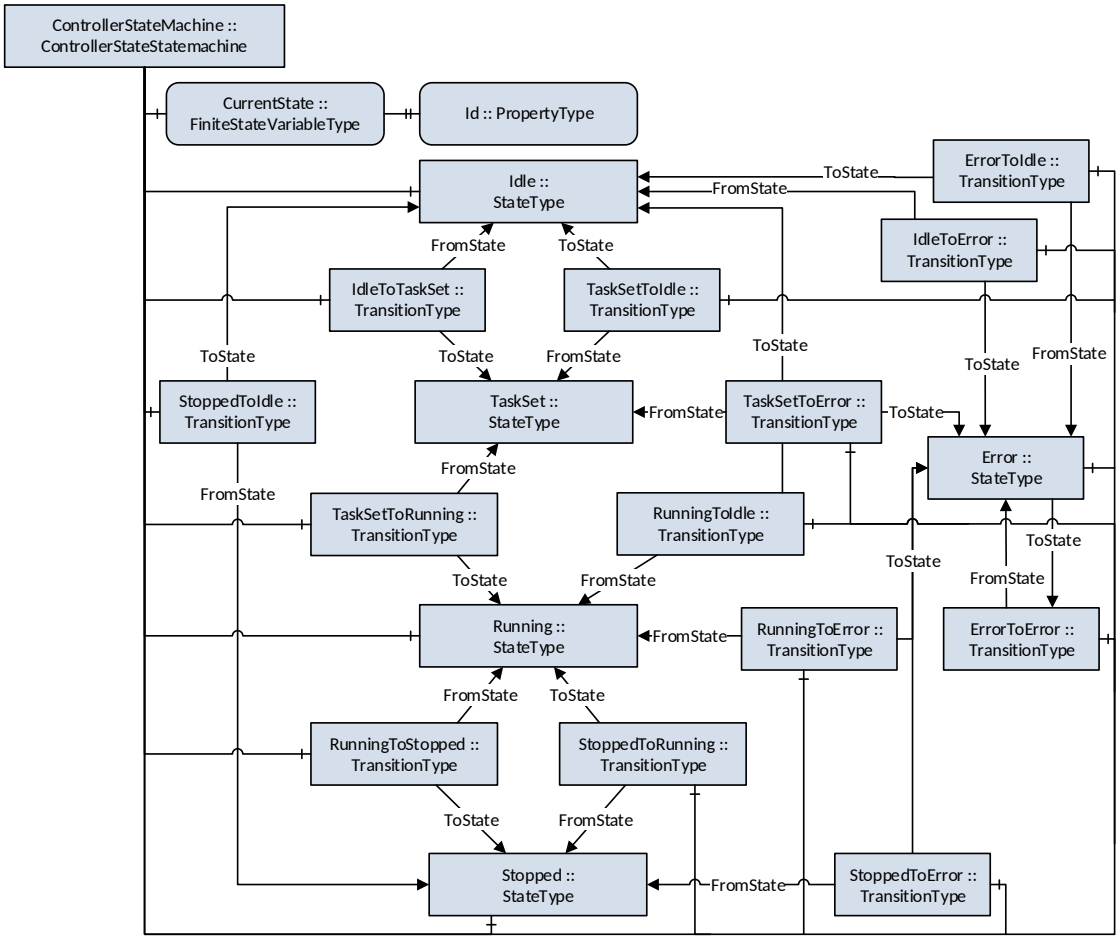

Abbildung 7.11: OPC UA Informationsmodell für die Controller StateMachine

KAPITEL 8

Validierung

Ziel des in der Arbeit vorgestellten Ansatzes OPC UA Information Model Design ist es, die Aufwände für die Modellierung von OPC UA Informationsmodellen zu reduzieren. Dies bedeute wiederum, dass die Komplexität der UML Modelle geringer sein sollte als die von OPC UA Modellen.
Für die Bewertung von Modellen werden dazu sogenannte Metriken eingesetzt. IEEE 1061 [133] definiert Metriken als:

> „Eine Softwarequalitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit."

Aktuell gibt es eine Vielzahl von Bewertungssätzen für UML Diagramme. Für OPC UA Modelle sind kaum Metriken vorhanden. Im Folgenden werden einige Methoden näher beschrieben und die erstellten Modelle damit validiert.

Als erste Metrik wird die visuelle Größe als Bewertungskriterium herangezogen. Diese Kennzahl liefert ein Maß für die Kompaktheit und Übersichtlichkeit des Designs.
Danach wird die auf die Bewertung von Bansiya und Davis [10], [38] eingegangen. Sie schlagen eine Methode zur Bewertung von UML-Klassendiagrammen vor. Die Quality of Object Oriented Design (QMOOD) bietet eine Möglichkeit, wie die Stabilität und Designqualität von Modelle identifiziert und charakterisiert werden kann. Diese Methode wird auch für das OPC UA Informationsmodell definiert und anschließend das Ergebnis gegenübergestellt [38].
Anschließend werden einige Kenngrößen für UML Klassendiagramme nach Genero [36] berechnet und auf OPC UA Informationsmodelle umgelegt und die Ergebnisse diskutiert.
Abschließend wird eine Bewertung der Zustandsdiagramme von UML und OPC UA nach einigen Kenngrößen nach Genero vorgenommen.

8.1 Visuelle Größe für Klassendiagramme

Van Elsuwe und Schmedding [142] bieten ebenfalls eine interessante Kennzahl, die visuelle Größe Die visuelle Größe eines Klassendiagramms berechnet sich laut [142] nach Formel 8.1. Es ist die Summe der Klassen, Schnittstellen, Attribute und Methoden.

$$VIS_{KlassenDiagr} = Anz_{Klassen} + Anz_{Schnittstellen} + Anz_{Attribute} + Anz_{Methoden} \quad (8.1)$$

Analog kann die visuelle Größe von OPC UA Informationsmodellen mit der Formel 8.2 beschrieben werden. Es ist die Summe an ObjektTypen, Objekten, Methoden, Variablen und Properties inklusive deren Attribute.

$$\begin{aligned} VIS_{OPCUA} = Anz_{Objects} + Anz_{Objects.Attributes} + Anz_{Variables} + \\ Anz_{Variables.Attributes} + Anz_{Properties} + Anz_{Properties.Attributes} + Anz_{Methods} \end{aligned} \quad (8.2)$$

Tabelle 8.1 zeigt das Ergebnis der visuellen Größe. Daraus geht hervor, dass das OPC

Tabelle 8.1: Visuelle Größe des UML Klassendiagramms und des OPC UA Informationsmodells

UML		OPC UA Detailiert		OPC UA Einfach	
Anzahl der Klassen	18	Anzahl der Objekttypen	18	Anzahl der Objekttypen	18
		Anzahl der Attribute der Objekttypen	72		
		Anzahl der Objekte	18	Anzahl der Objekte	18
Anzahl der Schnittstellen	0	Anzahl der Attribute der Objekte	72		
Anzahl der Attribute	31	Anzahl der Variables	18	Anzahl der Variables	18
		Anzahl der Attribute von Variables	72		
		Anzahl der Properties	30	Anzahl der Properties	30
		Anzahl der Attribute von Properties	120		
Anzahl der Methoden	10	Anzahl der Methoden	10	Anzahl der Methoden	10
		Anzahl der Attribute der Methoden			
Visuelle Größe	**59**		**430**		**94**

UA Modell um das siebenfache größer ist als das entsprechende UML Modell. Selbst in der einfachen Darstellung von OPC UA ist es noch immer fast doppelt so groß als eine entsprechendes Klassendiagramm.
Dies basiert auf der Tatsache, dass in OPC UA die Attribute der Knoten immer angegeben werden müssen. Beim OPC UA Information Model Design werden viele Attribute automatisch durch die Transformation generiert.
Somit eignet sich das UML Modell deutlich besser für den Austausch der Informationen als das OPC UA Informationsmodell, da es alle Informationen in einer kompakten Form darstellt.

8.2 Strukturelle Komplexität

Genero et al. [36] präsentieren eine Anzahl von Metriken für die Bewertung von Klassendiagrammen. Tabelle 8.2 zeigt die Metriken und deren Interpretation für OPC UA Informationsmodelle.

Tabelle 8.2: Strukturelle Komplexität für UML Klassendiagramme und OPC UA Informationsmodelle

	UML	OPC UA
NUMBER OF ASSOCIATIONS (NAssoc)	Die Gesamtzahl aller Assoziationen	Die Gesamtzahl aller Referenzen (ohne HasSubType) im Informationsmodell
NUMBER OF AGGREGATION (NAgg)	Die Gesamtzahl der Aggregations- bzw. Compositionsbeziehungen innerhalb eines Klassendiagramms	Die Gesamtzahl der Organize und HasComponent Referenzen im Informationsmodell.
NUMBER OF GENERALISATIONS (NGen)	Die Gesamtzahl der Generalisierungsbeziehungen innerhalb eines Klassendiagramms	Anzahl der HasSubtype Referenzen in einem Informationsmodell.
NUMBER OF AGGREGATIONS HIERARCHIES (NAggH)	Die Gesamtzahl der Aggregationshierarchien (Ganzteilstrukturen) innerhalb eines Klassendiagramms.	Die Gesamtzahl der hierarchischen Ebenen innerhalb eines Informationsmodells. (HasComponent zwischen Objekten bzw Objekttypen)
NUMBER OF GENERALISATIONS HIERARCHIES (NGenH)	Die Gesamtzahl der Generalisierungshierarchien innerhalb eines Klassendiagramms.	Die Gesamtzahl der Generalisierungshierarchien innerhalb eines Informationsmodells.
MAXIMUM DIT (MaxDIT)	Es ist das Maximum der DIT (Depth of Inheritance Tree)-Werte für jede Klasse des Klassendiagramms.	DIT für jeden Objekttyp.
MAXIMUM HAGG (MaxHAgg)	Es ist das Maximum der erhaltenen HAgg-Werte (Weg von Root Klasse zu den Blättern) für jede Klasse des Klassendiagramms.	Maximum der HAgg-Werte für jeden Objekttyp in einem Informationsmodell.
NUMBER OF CLASSES (NC)	Die Gesamtzahl der Klassen.	Anzahl der Objekttypen.
NUMBER OF ATTRIBUTES (NA)	Die Gesamtzahl der Attribute.	Anzahl der Variablen und Properties.
NUMBER OF METHODS (NM)	Die Gesamtzahl der Methoden.	Anzahl der Methoden.

Tabelle 8.3 zeigt das Ergebnis für die entsprechenden Modelle des AGVs.
Bei den Werten, welche die Anzahl der Modellelement beschreibt, ist eine starke Abweichung zwischen UML und OPC UA zu beobachten. Dies beruht einerseits auf der Tatsache, dass im OPC UA Informationsmodell neben Typen auch deren Instanzen verwendet werden. Anderseits werden in der OPC UA Notation viele Attribute über eigene Knoten dargestellt, was somit die Anzahl der Elemente im Informationsmodell erhöht.

Die Kennzahlen, welche die Struktur der Modelle beschreiben, unterscheiden sich hingegen kaum. Dies kommt daher, dass die Metamodelle von UML und OPC UA ähnliche Mechanismen für die Strukturierung anbieten.

Tabelle 8.3: Strukturelle Komplexität für UML Klassendiagramme und OPC UA Informationsmodelle

	UML	OPC UA
NUMBER OF ASSOCIATIONS (NAssoc)	8	72
NUMBER OF AGGREGATION (NAgg)	8	44
NUMBER OF GENERALISATIONS (NGen)	9	9
NUMBER OF AGGREGATIONS HIERARCHIES (NAggH)	7	16
NUMBER OF GENERALISATIONS HIERARCHIES (NGenH)	3	3
MAXIMUM DIT (MaxDIT)	2	2
MAXIMUM HAGG (MaxHAgg)	3	6
NUMBER OF CLASSES (NC)	18	34
NUMBER OF ATTRIBUTES (NA)	31	48
NUMBER OF METHODS (NM)	10	10

8.3 Qualitätsmerkmale aus Designeigenschaften

Bansiya und Davis definieren in der QMOOD [10] sechs Qualitätsmerkmale, welche eine Gewichtung von anderen Metriken darstellen. Diese Attribute lauten *Reusability, Flexibility, Understandability, Functionality, Extendibility, Effectiveness*. Diese Faktoren beschreiben sehr gut die Qualität des Designs. Die einzelnen Faktoren sind nachfolgend beschrieben. Für die Ermittlung dieser zusammengesetzten Faktoren sind einige anderen Metriken notwendig. Diese werden nachfolgend erklärt und für das Beispiel berechnet.

Reusability

$$\begin{aligned} Reusability = -0,25 * Coupling + 0,25 * Cohesion + \\ 0,5 * Messaging + 0,5 * DesignSize \end{aligned} \tag{8.3}$$

Flexibility

$$\begin{aligned} Flexibility = 0.25 * Encapsulation - 0.25 * Coupling + \\ 0.5 * Composition + 0.5 * Polymorphism \end{aligned} \tag{8.4}$$

Understandability

$$\begin{aligned} Understandability = -0,33 * Abstraction + 0,33 * Encapsulation - \\ 0,33 * Coupling + 0,33 * Cohesion - 0,33 * Polymorphism - \\ 0,33 * Complexity - 0,33 * DesignSize \end{aligned} \tag{8.5}$$

Functionality

$$\begin{aligned} Functionality = 0,12 * Cohesion + 0,22 * Polymorphism + \\ 0,22 * Messaging + 0,22 * DesignSize + 0,22 * Hierarchies \end{aligned} \tag{8.6}$$

Abbildung 8.1: QMOOD (Quality Model for Object Oriented Design) für Klassendiagramme

	UML	OPC UA
Design size (DSC)	Gesamtzahl der Klassen im Design	Gesamtzahl der Objects und ObjectTypes
Hierarchies (NOH)	Gesamtzahl der Root Klassen im Design	Gesamtzahl ObjectTypes im Informationsmodell
Abstraction (ANA)	Durchschnittliche Anzahl von Klassen von der eine Klasse erbt	Durchschnittliche Anzahl von ObjektTypen von der ein Objekt bzw. Objekttyp erbt
Encapsulation (DAM)	Durchschnittliches Verhältnis der Anzahl der privaten Attribute zur Gesamtzahl der in der Klasse deklarierten Attribute.	Verhältnis der Variablen mit UserWriteMask bzw WriteMask Read zu denen mit Write
Coupling (DCC)	Durchschnittliche Anzahl von verschiedenen Klassen, mit denen eine Klasse direkt verbunden ist.	Anzahl der verschiedenen Knoten mit denen ein Objekt bzw Typ direkt verbunden ist.
Cohesion (CAM)	Verwandtschaft zwischen den Methoden einer Klasse auf der Grundlage der Parameterliste der Methoden.	Verwandtschaft zwischen zwei Methoden basierend auf den Input Arguments
Composition (MOA)	Anzahl der Teil-Ganz-Beziehung, die durch die Verwendung von Attributen realisiert wird	Anzahl der HasComponent bzw. Organize Referenzen
Inheritance (MFA)	Es ist das Verhältnis der Anzahl der von einer Klasse geerbten Methoden zu der Gesamtzahl der Methoden, die von Member-Methoden der Klasse zugegriffen werden können	Analog zur UML Metrik
Polymorphism (NOP)	Es zählt die Anzahl der Methoden, die ein polymorphes Verhalten aufweisen	Analog zur UML Metrik
Messaging (CIS)	Durchschnittliche Anzahl der öffentlichen Methoden in einer Klasse.	Durchschnittliche Gesamtzahl der Methoden mit UserWritemask Read eines Objektes bzw. ObjektTyps
Complexity (NOM)	Diese Metrik misst die Gesamtzahl der Methoden in einer Klasse.	Gesamtzahl der Methods im Informationsmodell

Extendibility

$$\begin{aligned} Extendibility = 0,5 * Abstraction - 0,5 * Coupling + \\ 0,5 * Inheritance + 0,5 * Polymorphism \end{aligned} \tag{8.7}$$

Effectiveness

$$\begin{aligned} Effectiveness = 0,2 * Abstraction + 0,2 * Encapsulation + \\ 0,2 * Composition + 0,2 * Inheritance + 0,2 * Polymorphism \end{aligned} \tag{8.8}$$

Tabelle 8.4 stellt das Ergebnis der QMOOD dar. Es zeigt die jeweiligen Kennzahlen in absoluten Zahlen und deren normierte Werte. Die Stärken von UML liegen in der

Tabelle 8.4: Ergebnis der Design Qualität für das FTS

	UML		OPC UA	
DSC(design size in classes)	18,0	1,00	34,00	1,89
NOH(number of hierarchies)	9,0	1,00	9,00	1,00
ANA (average number of ancestors)	0,5	1,00	0,26	0,53
DAM(data access metrics)	0,5	1,00	0,54	1,00
DCC(direct class coupling)	0,9	1,00	1,71	1,81
CAM(cohesion among methods of class)	0,0	1,00	0,00	1,00
MOA(measure of aggregation)	8,0	1,00	19,00	2,38
MFA(measure of functional abstraction)	0,0	1,00	0,00	1,00
NOP(number of polymorphic methods)	0,0	1,00	0,00	1,00
CIS(class interface size)	0,6	1,00	0,29	0,53
NOM(number of methods)	13,0	1,00	13,00	1,00
Reusability	9,0	1,00	16,72	1,01
Flexibility	3,9	1,00	9,21	1,49
Understandability	-10,5	-0,99	-15,98	-1,39
Functionality	6,1	1,00	9,52	1,09
Extendibility	-0,2	1,00	-0,72	0,36
Effectiveness	1,8	1,00	3,96	1,18
TQI (Total Quality Index)	4,01		3,73	

Verständlichkeit des Designs. OPC UA punktet wiederum bei der Flexibilität. Die Werte Wiederverwendbarkeit, Erweiterbarkeit und Effektivität sind bei beiden Designs nahezu ident. Dieses Ergebnis bestätigt auch die Verwendung von UML für die Informationsmodellierung, da es benutzerfreundlicher und einfacher zu verwenden ist.

Zusammenfassend wird ein Qualitätswert (TQI) ermittelt. Dieser ist bei UML insgesamt höher, was ebenfalls für UML als Modellierungssprache für das OPC UA Information Model Design spricht.

8.4 Size Metrics für UML Zustandsdiagramme

Für die Validierung der UML Zustandsdiagramme bzw. OPC UA StateMachines werden einige Metriken herangezogen und miteinander verglichen. Genero et al. [35] und Cruz et al. [26] definieren eine Anzahl von Metriken, welche in Tabelle 8.5 dargestellt sind.

In Tabelle 8.5 ist ersichtlich, dass die Kennzahlen für OPC UA und UML nahezu ident sind. Dies zeigt, dass UML und OPC UA dieselben Modellelemente für die Modellierung von Zustandsautomaten besitzen. Der große Unterschied besteht nur bei der Repräsentation von Transitionen. Diese können in UML einfach dargestellt werden. In OPC UA

Name	UML		OPC UA	
NSS	Anzahl der simplen Zustände	5	Anzahl der State Objekte	5
NCS	Anzahl der zusammengesetzten Zustände	0	Anzahl SubStateMachines	0
NG	Anzahl der Guards	0	Anzahl der Guard Objekte	0
NT	Anzahl der Transitionen	11	Anzahl der Transition Objekte + Anzahl der Referenezen	39

Tabelle 8.5: Metriken für die Bewertung von State Machines

entsprechen Transitionen Objekten, welche mittels Referenzen miteinander verbunden sind. Daher ist hier die Anzahl dreimal so hoch.

8.5 Bewertung des OPC UA Information Model Design

Dieses Kapitel hat unterschiedliche Bewertungskriterien für OPC UA Information Model Design präsentiert. Zusammenfassend kann gesagt werden, dass UML für die Darstellung von Informationen für Information Interfaces im Vergleich zu OPC UA besser geeignet ist und können einfacher ausgetauscht werden. Dies Hilft bei der Erstellung von Modelle und vereinfacht den Prozess.

Die strukturelle Komplexität von UML ist geringer als die von OPC UA. Dies ist der Tatsache geschuldet, dass in OPC UA die Informationen in Form eines Graphen bestehend aus Knoten und Referenzen abgebildet sind.

Anders sieht es hingegen bei den Designeigenschaften aus. Hier ist UML bei der Verständlichkeit besser als OPC UA. OPC UA bietet auf Grund seines knotenbasierten Modells eine höhere Flexiblität und die Modelle können leichter wiederverwendet werden. Die Funktionalität, Erweiterbarkeit und Effektivität sind bei beiden Modellierungssprachen ähnlich.
Der Total Quality Index als Maßstab für die Designqualität ist bei UML ein wenig höher.

Bei den Zustandsdiagrammen wurden Size Metriken zur Bewertung verwendet. Da OPC UA durch den Guarded State Machines Ansatz erweitert wurde, gibt es hier kaum Unterschiede zwischen den beiden Modellierungssprachen. Lediglich die Anzahl der Referenzen zwischen Objekten ist in OPC UA höher.

Zusammenfassend bietet UML den erwarteten Vorteil, dass es (i) einfacher zu verwenden, (ii) leichter verständlich, und (iii) gut für die abstrakte Modellierung von OPC UA Informationsmodellen geeignet ist.

KAPITEL 9

Zusammenfassung und Ausblick

Die Arbeit liefert einen Überblick über den Digitalisierungstrend in der Fertigungstechnik. Ausgehend von Konzepten wie CPS und CPPS sowie ersten Umsetzungen in Form von der I4.0 Komponente wird der Einfluss von Informations- und Kommunikationstechnologien gezeigt. Diese Konzepte sind durch eine enge Vernetzung untereinander charakterisiert.

Außerdem wird ein Überblick über diese M2M Kommunikationslösungen in Kapitel 3 im Bereich der Fertigungstechnik gegeben. Die aktuell verfügbaren Lösungen OPC UA, MTConnect und MQTT sowie deren Vor- und Nachteile werden gegenübergestellt.

Kapitel 4 liefert einen Überblick über modellgetriebene Softwareentwicklung und geht im speziellen auf den von der OMG standardisierten Model Driven Architecture (MDA) Ansatz ein. Außerdem werden die Modelle, Metamodelle und Notationen von UML und OPC UA aufgelistet und durch die Ansätze von AutomationML und MTConnect ergänzt.

In Kapitel 5 wird das OPC UA Information Model Design auf Basis des MDA Ansatzes erklärt und die einzelnen Phasen und deren Modelle erläutert. Ergänzt wird dieses Kapitel durch das Kapitel 6, welches die Transformation von UML zu OPC UA beschreibt.
Neben den Transformationsregeln von UML in OPC UA wird ein UML Profil für die Erweiterung von UML präsentiert. Additionell wird eine Möglichkeit vorgestellt, OPC UA so zu erweitern, dass es Fähigkeiten von UML bekommt. Dies wird am Beispiel von Guards für Zustandsdiagramme gezeigt.

Die Metriken in Kapitel 8 zeigen die Stärken von UML Diagrammen für den Designprozess von OPC UA Informationsmodellen auf. Vor allem für die Beschreibung der statischen Struktur stellt UML einen Mehrwert dar. Sowohl bei der Größe, der strukturellen Komplexität sowie der Designqualität zeigt UML seine Vorteile.

Der in dieser Arbeit vorgestellte Ansatz und prototypische Implementierung soll weiter

verfolgt werden.
Modeling Tools for OPC UA (model-UA)[1] ist ein Open Source Projekt, das Entwickler bei der Erstellung von OPC UA Informationsmodellen unterstützt. Obwohl es zahlreiche Open Source Implementierungen von OPC UA Software-Stacks gibt, gibt es kaum Open Source Tool für die OPC UA Informationsmodellierung. model-UA zielt darauf ab, dies durch die Bereitstellung eines grafischen Editors für die Erstellung von OPC UA Informationsmodellen zu ändern. Außerdem bietet es eine Sammlung von Plugins, um die Modelle in OPC UA-konforme XML-Dateien zu exportieren. Dieser Export basiert auf den Überlegungen, welcher in dierser Arbeit durchgeführt wurden. Die damit erstellten Dateien dienen als Input für jede OPC UA Implementierung, egal ob kommerziell oder Open Source.

[1] https://github.com/model-UA

Literaturverzeichnis

[1] P. Adolphs, S. Auer, H. Bedenbender, M. Billmann, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele-Dunsche, et al. Struktur der verwaltungsschale: Fortentwicklung des referenzmodells für die industrie 4.0-komponente. *Bundesministerium für Wirtschaft und Energie (BMWi). Berlin: Spreedruck Berlin GmbH*, 2016.

[2] S. Alhir. Understanding the model driven architecture (mda). *Methods & Tools*, 11(3):17–24, 2003.

[3] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41, 2003.

[4] ATLAS Group. ATL: Atlas Transformation Language. Technical Report January, ATLAS Group, 2006.

[5] AutomationML e.V. Whitepaper AutomationML Part 1 -Architecture and general requirements. Technical report, AutomationML e.V, 2014.

[6] R. Baheti and H. Gill. Cyber-physical systems. *The impact of control technology*, 12:161–166, 2011.

[7] H. Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb.* Springer-Verlag, 2011.

[8] S. Bandyopadhyay and A. Bhattacharyya. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, page 334–340. IEEE, 2013.

[9] M. Banes and E. Finch. Collada-digital asset schema release 1.5. 0 specification. *Khronos Group, Sony Computer Entertainment Inc*, 2008.

[10] J. Bansiya and C.G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, 28(1):4–17, 2002.

[11] T. Bauernhansl, M. Ten Hompel, and B. Vogel-Heuser. *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung, Technologien und Migration.* Springer, 2014.

[12] L. Berardinelli, A. Mazak, O. Alt, M. Wimmer, and G. Kappel. Model-Driven Systems Engineering: Principles and Application in the CPPS Domain. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, page 261–299. Springer International Publishing, Cham, 2017.

[13] K. Bettenhausen and S. Kowalewski. Cyber-physical systems: Chancen und nutzen aus sicht der automation. *VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik*, 2013.

[14] H. Bonin. *Systemanalyse für Softwaresysteme.* Universität, 2006.

[15] G. Booch. *Object Oriented Analysis and Design with Applications*, volume 2. Addison-Wesley, 1994.

[16] G. Booch. *The unified modeling language user guide.* Pearson Education India, 2005.

[17] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[18] M. Broy. *Informatik Eine grundlegende Einführung: Band 1: Programmierung und Rechnerstrukturen*, volume 1. Springer-Verlag, 2013.

[19] A. Bröhl. *Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden.* Oldenbourg, 1993.

[20] J. Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[21] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, 2005.

[22] D. Calegari and N. Szasz. Verification of model transformations: A survey of the state-of-the-art. *Electronic Notes in Theoretical Computer Science*, 292:5–25, 2013.

[23] CEN-CENELEC-ETSI Smart Grid Coordination Group. Smart grid reference architecture. Technical report, CEN-CENELEC-ETSI Smart Grid Coordination Group, 2012.

[24] P. Coad and E. Yourdon. *OOA: objektorientierte Analyse.* Prentice Hall, 1994.

[25] V. Cortellessa, A. Di Marco, and P. Inverardi. Software performance model-driven architecture. In *Proceedings of the 2006 ACM symposium on Applied computing*, page 1218–1223. ACM, 2006.

[26] J. Cruz-Lemus, M. Genero, and M. Piattini. Metrics for uml statechart diagrams. In *Metrics for Software Conceptual Models*, page 237–272. World Scientific, 2005.

[27] A. Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[28] Die SOPHISTen. Anforderungen und Architekturen für komplexe Systeme, 2017.

[29] R. Drath, A. Lüder, J. Peschke, and L. Hundt. AutomationML - The glue for seamless Automation engineering. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, page 616–623, 2008.

[30] D. Edstrom. *MTConnect: to measure is to know.* Virtual Photons Electrons, LLC Ashburn, VA, 2013.

[31] H. Elshaafi, M. Vinyals, M. Dibley, I. Grimaldi, and M. Sisinni. Combination of standards to support flexibility management in the smart grid, challenges and opportunities. In *Smart Grid Inspired Future Technologies*, page 143–151. Springer, 2017.

[32] Fieldbus Foundation. Field Device Integration (FDI) – Part 5: FDI Information Model, Version 1.1.0. Technical report, Fieldbus Foundation, 2017.

[33] T. Frühwirth, F. Pauker, A. Fernbach, I. Ayatollahi, W. Kastner, and B. Kittl. Guarded state machines in OPC UA. In *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2015.

[34] E. Geisberger and M. Broy, editors. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems.* acatech STUDIE. Springer-Verlag, Berlin Heidelberg, 2012.

[35] M. Genero, D. Miranda, and M. Piattini. Defining and validating metrics for uml statechart diagrams. *Proceedings of QAOOSE*, 2002, 2002.

[36] M. Genero, M. Piattini, and C. Calero. Empirical validation of class diagram metrics. *ISESE 2002 - Proceedings, 2002 International Symposium on Empirical Software Engineering*, page 195–203, 2002.

[37] T. Gherbi, D. Meslati, and I. Borne. Mde between promises and challenges. In *Computer Modelling and Simulation, 2009. UKSIM'09. 11th International Conference on*, page 152–155. IEEE, 2009.

[38] P. Goyal and G. Joshi. Qmood metric sets to assess quality of java program. In *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, page 520–533. IEEE, 2014.

[39] V. Gruhn, D. Pieper, and C. Röttgers. *MDA®: Effektives Software-Engineering mit UML2® und EclipseTM.* Springer-Verlag, 2007.

[40] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[41] J. Hohmann. Cyber-Physische Systeme – RobIN 4.0, 2017.

[42] S. Hu. Paradigms of manufacturing—a panel discussion. In *3rd Conference on reconfigurable manufacturing, Ann Arbor, Michigan, USA*, 2005.

[43] IEC. IEC 7498-1: Information technology-open systems interconnection-basic reference model: The basic model. Technical report, IEC, 1994.

[44] IEC. IEC 19501 - Informationstechnik: Offene verteilte Verarbeitung - Vereinheitlichte Modellierungssprache (UML) Version 1.4.2. Technical Report 2, IEC, 2005.

[45] IEC. Iec 62424- representation of process control engineering–requests in pi diagrams and data exchange between pid tools and pce-cae tools. Technical report, IEC, 2008.

[46] IEC. IEC 62264 - Enterprise-control system integration. Technical report, IEC, 2013.

[47] IEC. IEC 62541 - OPC Unified Architecture. Technical report, IEC, 2015.

[48] International Electrotechnical Commision et al. IEC 62890 life-cycle management for systems and products used in industrial-process measurement, control and automation. *committee Draft*, page 10–17, 2014.

[49] ISO. ISO 639 - Language Codes. Technical report, ISO, 2002.

[50] ISO. Iso 13399-1: Overview, fundamental principles and general information model. Technical report, ISO, 2014.

[51] ISO/IEC. ISO/IEC 20922 - Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1. Technical report, ISO/IEC, 2016.

[52] I. Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach: A Use CASE Approach.* Addison-Wesley Professional, New York : Wokingham, Eng. ; Reading, Mass, 01 edition, June 1992.

[53] J. Jasperneite, A. Neumann, and F. Pethig. OPC UA versus MTConnect. *Computer & Automation*, page 16–21, 2015.

[54] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, jun 2008.

[55] H. Kagermann, W. Wahlster, and J. Helbig. Umsetzungsempfehlungen für das zukunftsprojekt industrie 4.0. *Abschlussbericht des Arbeitskreises Industrie*, 4, 2013.

[56] G. Kappel, M. Seidl, C. Huemer, and M. Brandsteidl. *UML@ Classroom: Eine Einführung in die objektorientierte Modellierung.* dpunkt. verlag, 2012.

[57] M. Keinert. Industrielle kommunikation: OPC UA - eine standortbestimmung, 05 2013.

[58] M. Kempa and Z. Mann. Model driven architecture. *Informatik-Spektrum*, 28(4):298–302, 2005.

[59] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[60] A. Kleppe, J. Warmer, and W. Bast. Mda explained. the practice and promise of the model driven architecture. *ed: Addison Wesley Reading*, 2003.

[61] Y. Koren. *The global manufacturing revolution: product-process-business integration and reconfigurable systems*, volume 80. John Wiley & Sons, 2010.

[62] A. Kriouile, T. Gadi, and Y. Balouki. Cim to pim transformation: A criteria based evaluation. *International Journal of Computer Technology and Applications*, 4(4):616, 2013.

[63] S. Lass and D. Kotarski. It-sicherheit als besondere herausforderung von industrie 4.0. *Industrie*, 4:397–419, 2014.

[64] B. Lee, D. Kim, H. Yang, and S. Oh. Model transformation between opc ua and uml. *Computer Standards & Interfaces*, 50:236–250, 2017.

[65] E. Lee. Cyber physical systems: Design challenges. *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, page 363–369, 2008.

[66] J. Lee, B. Bagheri, and H. Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[67] D. Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[68] Y. Lu, K. Morris, and S. Frechette. Current standards landscape for smart manufacturing systems. *National Institute of Standards and Technology, NISTIR*, 8107:22–28, 2016.

[69] A. Lüder and N. Schmidt. AutomationML in a Nutshell. In *Handbuch Industrie 4.0 Bd.2*, page 213–258. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

[70] N. Mahalik. *Fieldbus technology: industrial network standards for real-time distributed control*. Springer Science & Business Media, 2013.

[71] W. Mahnke and S. Leitner. Opc unified architecture-the future standard for communication and information modeling in automation. *ABB Review*, 3:2009, 2009.

[72] W. Mahnke, S. Leitner, and M. Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.

[73] I. Menken. *Model-Driven Architecture Complete Certification Kit - Core Series for IT*. Emereo Publishing, 2013.

[74] L. Monostori. Cyber-physical production systems: Roots, expectations and r&d challenges. *Procedia CIRP*, 17:9–13, 2014.

[75] MTConnect Institute. MTConnect Standard Part 1 - Overview and Protocol Version 1.3.0. Technical report, MTConnect Institute, 09 2014.

[76] MTConnect Institute. MTConnect ® Standard Part 1 -Overview and Protocol Version 1.3.0. Technical report, MTConnect Institute, 2014.

[77] MTConnect Institute. MTConnect® Standard Part 2 – Device Information Model Version 1.3.0. Technical report, MTConnect Institute, 2014.

[78] MTConnect Institute. MTConnect® Standard Part 3 – Streams, Events, Samples, and Condition Version 1.3.0. Technical report, MTConnect Institute, 2014.

[79] MTConnect Institute. MTConnect® Standard Part 3.1 – Interfaces Version 1.3.0. Technical report, MTConnect Institute, 2014.

[80] MTConnect Institute. MTConnect® Standard Part 4.0 – Assets Version 1.3.0. Technical report, MTConnect Institute, 2014.

[81] MTConnect Institute. MTConnect® Standard Part 4.1 – Cutting Tools Version 1.3.0. Technical report, MTConnect Institute, 2014.

[82] MTConnect-Institute and OPC-Foundation. MTConnect-OPC UA Companion Specification - Version 1.0. Technical report, MTConnect-Institute and OPC-Foundation, 2012.

[83] N. Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, page 1–7. IEEE, 2017.

[84] OASIS. MQTT Version 3.1.1. Technical Report October, OASIS, 2014.

[85] Object Management Group. Model Driven Architecture (MDA). Technical Report June, Object Management Group, 2003.

[86] Object Management Group. Model Driven Architecture (MDA). Technical Report MDA Guide rev. 2.0, Object Management Group, 2014.

[87] Object Management Group. Object Constraint Language. Technical Report May, Object Management Group, 2014.

[88] Object Management Group. Omg mda guide rev. 2.0. Technical report, Object Management Group, 2014.

[89] Object Management Group. OMG Meta Object Facility (MOF) Core Specification - Version 2.5. Technical report, Object Management Group, 2015.

[90] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.5. Technical report, Object Management Group, 2015.

[91] Object Management Group. XML Metadata Interchange (XMI) Specification - Version 2.5.1. Technical report, Object Management Group, 2015.

[92] OPC Foundation. Data Access Custom Interface Standard Version 3.00. Technical report, OPC Foundation, 2003.

[93] OPC Foundation. OPC Unified Architecture for Devices (DI), Version 1.01. Technical report, OPC Foundation, 2013.

[94] OPC Foundation. OPC Unified Architecture for ISA-95 Common Object Model - Version 1.0. Technical report, OPC Foundation, 2013.

[95] OPC Foundation. OPC UA Information Model for AutomationML, Version 1.0.0. Technical report, OPC Foundation, 2015.

[96] OPC Foundation. OPC Unified Architecture - Wegbereiter der 4. industriellen (R)Evolution. Technical report, OPC Foundation, 2015.

[97] OPC Foundation. OPC Unified Architecture- Interoperabilität für Industrie 4.0 und das Internet der Dinge. Technical report, OPC Foundation, 2015.

[98] OPC Foundation. OPC Unified Architecture for Analyser Devices Companion Specification Release 1.1a. Technical report, OPC Foundation, 2015.

[99] OPC Foundation. OPC Unified Architecture Part 13: Aggregates Version 1.03. Technical report, OPC Foundation, 2015.

[100] OPC Foundation. OPC Unified Architecture Part 3: Address Space Model Version 1.03. Technical report, OPC Foundation, 2015.

[101] OPC Foundation. OPC Unified Architecture Part 4: Services Version 1.03. Technical report, OPC Foundation, 2015.

[102] OPC Foundation. OPC Unified Architecture Part 5: Information Model Version 1.03. Technical report, OPC Foundation, 2015.

[103] OPC Foundation. OPC Unified Architecture Part 6: Mappings Version 1.03. Technical report, OPC Foundation, 2015.

[104] OPC Foundation. OPC Unified Architecture Specification Part 1: Overview and Concepts Version 1.03. Technical report, OPC Foundation, 2015.

[105] OPC Foundation. OPC Unified Architecture Specification Part 10: Programs Version 1.02. Technical report, OPC Foundation, 2015.

[106] OPC Foundation. OPC Unified Architecture Specification Part 11: Historical Access Version 1.02. Technical report, OPC Foundation, 2015.

[107] OPC Foundation. OPC Unified Architecture Specification Part 8: Data Access Version 1.02. Technical report, OPC Foundation, 2015.

[108] OPC Foundation. OPC Unified Architecture Specification Part 9: Alarms and Conditions Version1.02. Technical report, OPC Foundation, 2015.

[109] OPC Foundation. OPC Unified Architecture - Interoperabilität für Industrie 4.0 und das Internet der Dinge, 2016.

[110] OPC Foundation. OPC Unified Architecture Specification Part 14: PubSub Version 1.04. Technical report, OPC Foundation, 2018.

[111] OPC Foundation and PLCopen. OPC UA Information Model for IEC 61131-3 - Release 1.00. Technical report, OPC Foundation and PLCopen, 2010.

[112] R. Oppliger. *IT-Sicherheit: Grundlagen und Umsetzung in der Praxis*. DuD-Fachbeiträge. Vieweg+Teubner Verlag, 2013.

[113] F. Pauker, T. Frühwirth, B. Kittl, and W. Kastner. A Systematic Approach to OPC UA Information Model Design. In *Procedia CIRP*, volume 57, 2016.

[114] F. Pauker, S. Wolny, M. Solmaz, and M. Wimmer. UML2opc-UATransforming UML Class Diagrams to OPC UA Information Models. *Procedia CIRP*, 67:128–133, January 2018.

[115] P. Pin-Shan Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[116] PLCopen. XML Formats for IEC 61131-3. Technical report, PLCopen, 2004.

[117] J. Raasch. *Systementwicklung mit strukturierten Methoden: ein Leitfaden für Praxis und Studium*. C. Hanser, 1993.

[118] S. Rohjans, K. Piech, and S. Lehnhoff. Uml-based modeling of opc ua address spaces for power systems. In *Intelligent Energy Systems (IWIES), 2013 IEEE International Workshop on*, page 209–214. IEEE, 2013.

[119] S. Rohjans, K. Piech, M. Uslar, and J. Cabadi. Cimbat-automated generation of cim-based opc ua-address spaces. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, page 416–421. IEEE, 2011.

[120] A. Roth. *Einführung und Umsetzung von Industrie 4.0: Grundlagen, Vorgehensmodell und Use Cases aus der Praxis.* Springer-Verlag, 2016.

[121] W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, page 328–338. IEEE Computer Society Press, 1987.

[122] J. Rumbaugh, M. Blaha, W. Premerlani, and F. Eddy. *Object-Oriented Modeling and Design.* Pearson, Englewood Cliffs, N.J, united states ed edition, 1991.

[123] C. Rupp et al. *Systemanalyse kompakt.* Springer-Verlag, 2013.

[124] C. Rupp et al. *Requirements-Engineering und-Management: Aus der Praxis von klassisch bis agil.* Carl Hanser Verlag GmbH Co KG, 2014.

[125] C. Rupp, S. Queins, et al. *UML 2 glasklar: Praxiswissen für die UML-Modellierung.* Carl Hanser Verlag GmbH Co KG, 2012.

[126] D. Sahl. Industriepolitik für Europa – Die EU als Standort industrieller Wertschöpfung zukunftsfähig machen, 2015.

[127] A. Scheer. Industrie 4.0: Wie sehen Produktionsprozesse im Jahr 2020 aus. *IMC AG*, 2013.

[128] M. Schleipen, S. Gilani, T. Bischoff, and J. Pfrommer. Opc ua & industrie 4.0-enabling technology with high diversity and variability. *Procedia CIRP*, 57:315–320, 2016.

[129] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(February):25–31, 2006.

[130] B. Schätz. The role of models in engineering of cyber-physical systems–challenges and possibilities. In *Tagungsband des Dagstuhl-Workshops*, page 91, 2014.

[131] B. Selić and S. Gérard. An Introduction to UML Profiles. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE*, V(2):27–43, 2014.

[132] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5):42–45, 2003.

[133] Software Engineering Standards Committee et al. Ieee standard for a software quality metrics methodology, std. 1061-1998. Technical report, Software Engineering Standards Committee, 1998.

[134] H. Stachowiak. *Allgemeine Modelltheorie.* Springer, Wien, 1973.

[135] T. Stahl, S. Efftinge, A. Haase, and M. Völter. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management.* dpunkt. verlag, 2012.

[136] T. Stahl, M. Völter, et al. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management.* dpunkt, Heidelberg, 2., aktualis. u. erw. aufl. edition, 2007.

[137] M. Stopper and B. Katalinic. Service-oriented architecture design aspects of opc ua for industrial applications. In *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, volume 2. Citeseer, 2009.

[138] The Association For Manufacturing Technology. Getting Started with MTConnect : Monitoring Your Shop Floor – What ' s In It For You ? Technical report, The Association For Manufacturing Technology, 2013.

[139] T. Trautner. Sensor- und steuerungsintegration mit mtconnect. Masterthesis, TU Wien, Wien, 2016.

[140] T. Trautner, F. Pauker, and B. Kittl. Advanced mtconnect asset management (amam). In *International Conference on Innovative Technologies*, 2016.

[141] F. Truyen. The fast guide to model driven architecture the basics of model driven architecture. *Cephas Consulting Corp*, 2006.

[142] H. Van Elsuwe and D. Schmedding. Metriken für UML-Modelle. *Informatik - Forschung und Entwicklung*, 18(1):22–31, 2003.

[143] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik. Industrie 4.0 Statusreport. Technical Report April, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, 2014.

[144] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik. Status Report-Reference Architecture Model Industrie 4.0 (RAMI4.0). Technical report, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, 2015.

[145] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik. Status Report-Reference Architecture Model Industrie 4.0 (RAMI4.0). Technical report, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, 2015.

[146] VDW and OPC Foundation. OPC UA Information Model for CNC Systems, Version 1.0. Technical report, VDW and OPC Foundation, 2017.

[147] G. Versteegen, A. Chughtai, H. Dörnemann, R. Heinold, R. Hubert, K. Salomon, and O. Vogel. *Software Management: Beherrschung des Lifecycles.* Xpert.press. Springer Berlin Heidelberg, 2013.

[148] B. Vogel-Heuser, C. Diedrich, and M. Broy. Anforderungen an cps aus sicht der automatisierungstechnik. *at–Automatisierungstechnik at–Automatisierungstechnik*, 61(10):669–676, 2013.

[149] V. Vyatkin. Software engineering in industrial automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 9(3):1234–1249, 2013.

[150] G. Wellenreuther and D. Zastrow. *Automatisieren mit SPS-Theorie und Praxis: programmieren mit STEP 7 und CoDeSys, Entwurfsverfahren, Bausteinbibliotheken; Beispiele für Steuerungen, Regelungen, Antriebe und Sicherheit; Kommunikation über AS-i-Bus, PROFIBUS, PROFINET, Ethernet-TCP/IP, OPC, WLAN; mit 108 Steuerungsbeispielen und 8 Projektierungen.* Springer Vieweg, 2015.

[151] K. Wiegers and J. Beatty. *Software requirements.* Pearson Education, 2013.

[152] ZVEI. Industrie4.0: Die Industrie 4.0-Komponente. Technical Report April, Zentralverband Elektrotechnik- und Elektronikindustrie e. V, 2015.

Acronyms

AGV Automated Guided Vehicle. 97, 99, 101, 104, 113

API Application Programming Interface. 57

ATL Atlas Transformation Language. viii, x, 32, 34, 93

AutomationML Automation Markup Language. 50–52, 54, 119

CIM Computation Independent Model. xii, 31, 56–59

CPPS Cyber-Physisches Produktions System. 2, 4, 5, 9, 11, 12, 51, 55, 119

CPS Cyber-Physical Systems. vii, ix, xiv, 2–11, 13, 53–56, 61, 63, 119

ERP Enterprise Resource Planning. 3

HMI Human Machine Interface. 19

HTTP Hypertext Transfer Protocol. 25

I4.0-Komponente Industrie 4.0-Komponente. 13, 14

IIoT Industrial Internet of Things. 2, 4

IKT Informations- und Kommunikationstechnologie. vii, ix, 2, 13

IoT Internet of Things. xi, 8, 10, 25

MDA Model Driven Architecture. viii, ix, 5–8, 29–31, 51, 54–58, 63, 119

MDE Model Driven Engineering. 29, 30, 32

MDSE Model-Driven Software Development. 66, 69

MES Manufacturing Execution System. 3

MOF Meta-Object Facility. 31, 58

MQTT Message Queue Telemetry Transport. xi, xiv, 8, 19, 25–27, 64

MTConnect MTConnect. 8, 23, 28, 47, 52, 64

OCL Object Constraint Language. 63, 76, 77, 84, 92

OMG Object Management Group. 30

OPC OLE for Process Control. 19

OPC Open Platform Communications. 3

OPC UA OPC Unified Architecture. vii–ix, xii, xiv, 3, 6–8, 15, 19–21, 27, 28, 41, 44, 45, 47, 52–56, 63, 64, 69, 70, 74, 78–80, 85, 88, 92, 117, 119

PIM Platform Independent Model. xiii, 7, 31, 54, 56–58, 63, 64, 70, 101

PSM Platform Specific Model. xii, 7, 31, 54, 56–58, 63, 64, 70

QMOOD Quality of Object Oriented Design. 111, 114, 115

R-PIM Restricted Platform Independent Model. xii, 57, 59, 61, 63, 70

RAMI 4.0 Referenzarchitekturmodell Industrie 4.0. xi, 4, 12–14, 53

SA strukturierte Analyse. 61

SCADA Supervisory Control and Data Acquisition. 19

SDK Software Development Kit. 53

SHDR Simple Hierarchical Data Representation. 24, 25

SOA Service Oriented Architecture. 21

UML Unified Modelling Language. 6–8, 34, 35, 52, 58, 59, 63, 64, 69, 70, 78, 80, 117, 119

XMI XML Metadata Interchange. 31